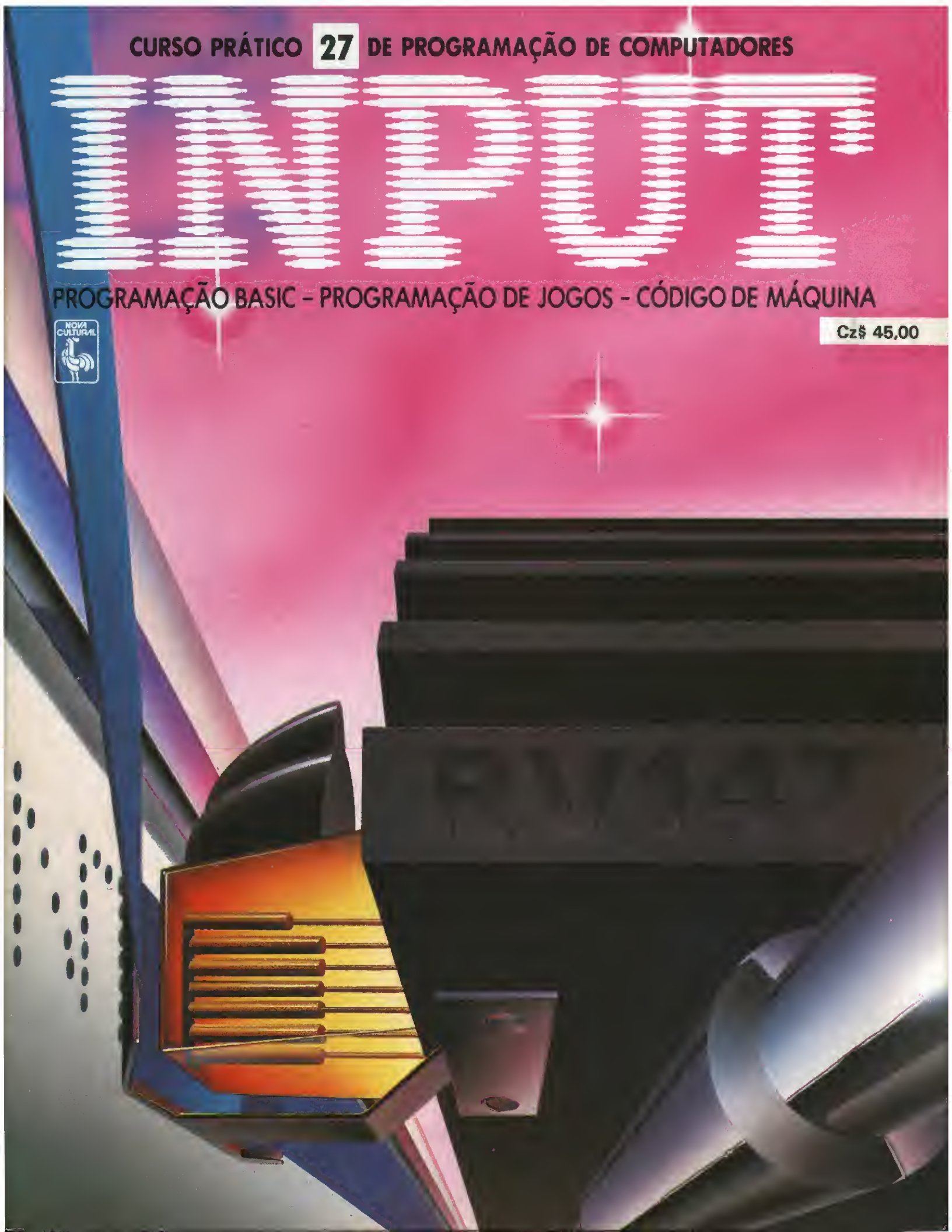
CURSO PRÁTICO 27 DE PROGRAMAÇÃO DE COMPUTADORES

INPUT

PROGRAMAÇÃO BASIC - PROGRAMAÇÃO DE JOGOS - CÓDIGO DE MÁQUINA

NOVA CULTURAL

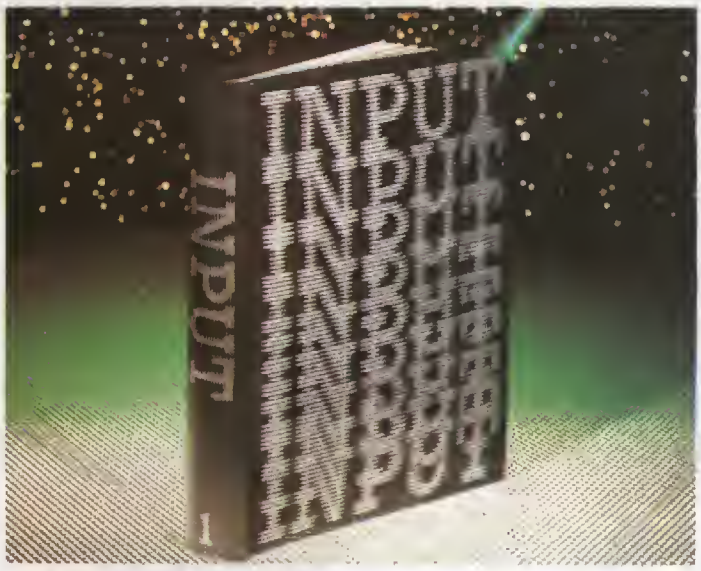
Cz$ 45,00



INPUT

Vol. 2 Nº 27

NESTE NÚMERO



PLANO DA OBRA
INPUT é uma obra editada em fascículos semanais, e cada conjunto de 15 fascículos compõe um volume. A capa para encadernação de cada volume estará à venda oportunamente.

FÉRIAS, VIAGENS, MUDANÇAS...
NÃO FIQUE COM A COLEÇÃO INCOMPLETA
Se você está saindo de férias, pretende viajar ou vai se ausentar por algum tempo, avise antecipadamente seu jornaleiro. Ele pode guardar os seus fascículos enquanto você estiver fora. Se, por qualquer motivo, você perdeu alguns números, peça-os também a seu jornaleiro, ou entre em contato com nossa Distribuidora:

1. **Pessoalmente** — Em *São Paulo*, os endereços são: rua Brigadeiro Tobias, 773, Centro; av. Industrial, 117, Santo André. No *Rio de Janeiro*, av. Mem de Sá, 191/193, Centro.
2. **Por carta** — Envie para:
   DINAP — Distribuidora Nacional de Publicações
   Números Atrasados
   Estrada Velha de Osasco, 132 — Jardim Teresa
   CEP 06040 — Osasco — SP
3. **Por telex** — Utilize o nº (011) 33 670 DNAP.

Em **Portugal**, os pedidos devem ser feitos à Distribuidora Jardim de Publicações Lda. — Qta. Pau Varais, Azinhaga de Fetais, 2685, Camarate, Lisboa; Apartado 57; Telex 43 069 JARLIS P.
Atenção: Após seis meses do encerramento da coleção, o atendimento dos pedidos dependerá da disponibilidade do estoque.
Obs.: Quando pedir livros, mencione sempre o título e/ou autor da obra, além do número da edição.

COLABORE CONOSCO
Encaminhe seus comentários, críticas, sugestões ou reclamações ao
SERVIÇO DE ATENDIMENTO AO LEITOR
Caixa Postal 9 442, São Paulo — SP.

EDITOR
RICHARD CIVITA
NOVA CULTURAL

Presidente
Flávio Barros Pinto
Diretoria
Carmo Chagas, Iara Rodrigues,
Pierluigi Bracco, Plácido Nicoletto,
Roberto Silveira, Shozi Ikeda,
Sônia Carvalho

REDAÇÃO
Diretor Editorial: Carmo Chagas
Editores Executivos:
Antonio José Filho, Berta Sztark Amar
Editor Chefe: Paulo de Almeida
Editoras Assistentes: Ana Lúcia B. de Lucena,
Marisa Soares de Andrade
Chefe de Arte: Carlos Luiz Batista
Assistentes de Arte: Dagmar Bastos Sampaio,
Grace Alonso Arruda, Monica Lenardon Corradi

Secretária de Redação/Coordenadora: Stefania Crema
Secretário de Redação: Mauro de Queiroz

Colaboradores
Consultor Editorial Responsável:
Dr. Renato M. E. Sabbatini
(Diretor do Núcleo de Informática Biomédica da
Universidade Estadual de Campinas-SP)
Execução Editorial: DATAQUEST Assessoria
em Informática Ltda., Campinas
Tradução, adaptação, programação e redação:
Abílio Pedro Neto, Aluísio J. Dornellas de Barros,
Marcelo R. Pires Therezo, Marcos Huascar Velasco,
Raul Neder Porrelli, Ricardo J. P. de Aquino Pereira
Coordenação Geral: Rejane Felizatti Sabbatini

COMERCIAL
Diretor Comercial: Roberto Silveira
Gerente Comercial: Joaquim Celestino da Silva
Gerente de Circulação: Denise Mozol
Gerente de Propaganda e Publicidade: José Carlos Madio
Gerente de Pesquisa e Análise de Mercado:
Wagner M. P. Nabuco de Araújo

CLC

A Editora Nova Cultural Ltda. é uma empresa do
Grupo CLC — Comunicações, Lazer, Cultura S.A.
Presidente: Richard Civita
Diretoria: Flávio Barros Pinto, João Gomez,
Menahen M. Politi, Renê C. X. Santos,
Stélio Alves Campos

Edição organizada pela Editora Nova Cultural Ltda.
Av. Brigadeiro Faria Lima, 2000 - 3º andar
CEP 01452 - São Paulo - SP - Brasil
(Artigo 15 da Lei 5 988, de 14/12/1973).
Esta obra foi composta pela AM Produções Gráficas Ltda.
e impressa pela Companhia Lithographica Ypiranga.

COMO ESCOLHER UMA IMPRESSORA

- ESCOLHA O TIPO CERTO DE IMPRESSORA
- PAPEL PARA IMPRESSÃO
- COMO CONECTAR A IMPRESSORA AO SEU COMPUTADOR

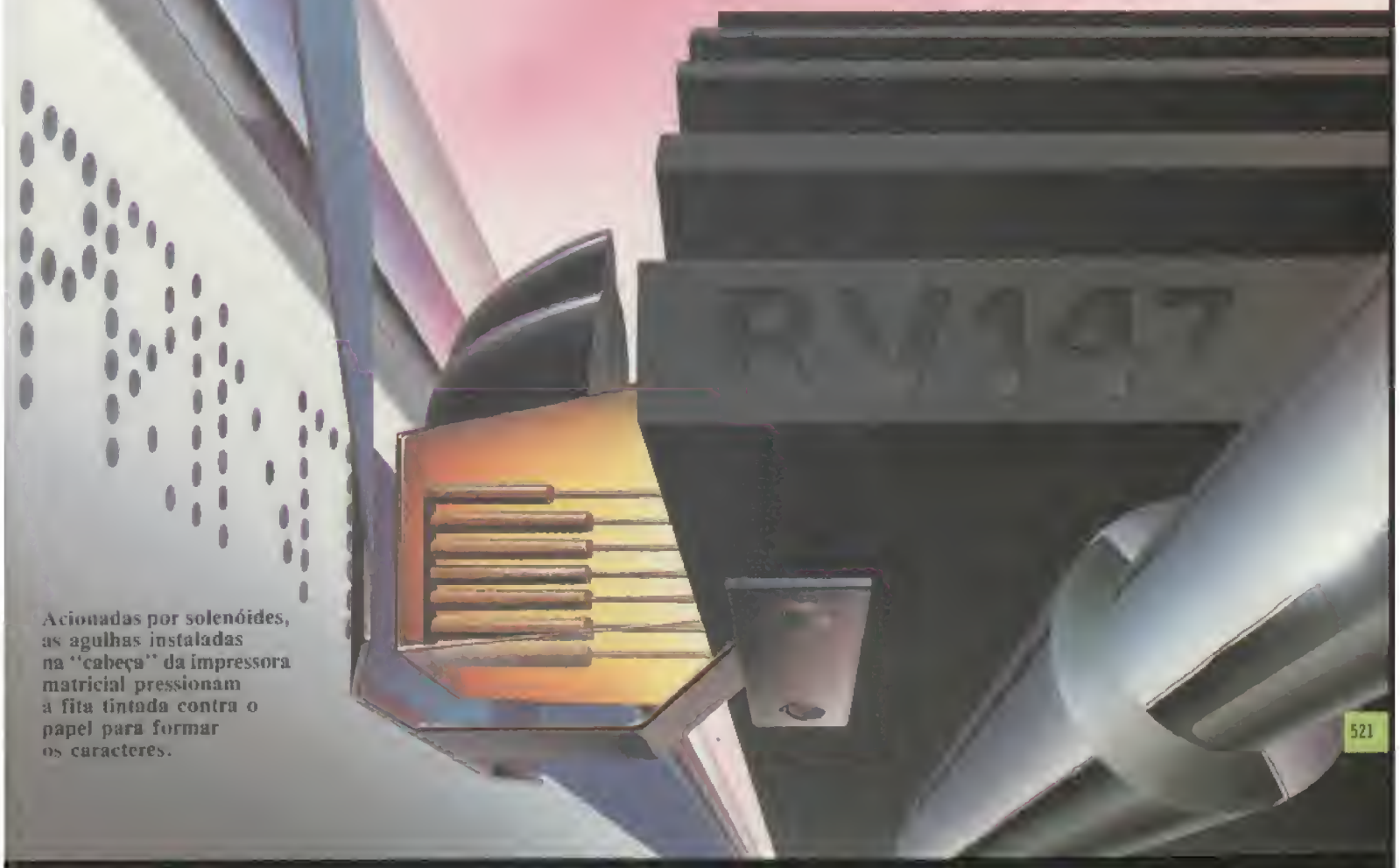
Se você já desejou algum dia uma listagem impressa de seus programas, ou uma cópia dos gráficos que aparecem na tela, chegou o momento de adquirir uma impressora.

Periférico dos mais úteis para um microcomputador, uma impressora contribui não só para a concretização de muitas aplicações novas, como também para um notável progresso na produtividade e na eficiência da programação. Entretanto, como ela representa um investimento considerável, e como existe no mercado uma grande variedade de tipos e preços, é necessário planejar com bastante cuidado a aquisição do modelo mais adequado.

Embora a impressora possa ser vista apenas como um periférico de saída alternativo para o vídeo, isto não quer dizer que ambos tenham as mesmas funções. O papel principal da impressora é assegurar um registro permanente de qualquer informação armazenada ou processada pelo computador, mesmo que esta nunca seja exibida na tela. Mas ela pode proporcionar também cópias da tela em papel (o que é conhecido como *hardcopy*, em jargão técnico).

Há muitos tipos de programas capazes de desenhar na tela gráficos científicos (chamados também de "arte computacional") que o usuário gostaria de copiar, arquivar, ou até colocar em uma moldura. O mesmo acontece com os desenhos técnicos produzidos no vídeo por software de projetos (CAD ou *Computer-Aided-Design*). Tudo isso pode ser reproduzido no papel por uma impressora gráfica adequada.

Certas aplicações, porém, exigem apenas uma impressora de textos; por exemplo, se você trabalha com programas de bancos de dados ou com progra-



Acionadas por solenóides, as agulhas instaladas na "cabeça" da impressora matricial pressionam a fita tintada contra o papel para formar os caracteres.

mas contábeis, que não necessitem de gráficos, pode encontrar uma impressora mais barata, e igualmente adequada.

Um computador doméstico pode ser usado de modo semelhante a uma máquina de escrever: seu teclado permite datilografar cartas, artigos, livros etc. Existem programas de processamento de textos que manipulam com rapidez e eficiência palavras, linhas e parágrafos. Nesse tipo de aplicação, a impressora é um periférico essencial.

A impressora funciona ainda como elemento de apoio para a atividade de programação. Neste particular, os méritos do vídeo são muitos, mas as pessoas costumam ter mais familiaridade com a palavra impressa. Por outro lado, a maioria dos programadores gosta de ter uma cópia impressa do programa; esta não só dá uma idéia global do programa, como facilita o trabalho de detecção e correção de erros. Além disso, é sempre bom contar com uma listagem do programa em papel para manter um arquivo paralelo.

TIPOS DE IMPRESSORA

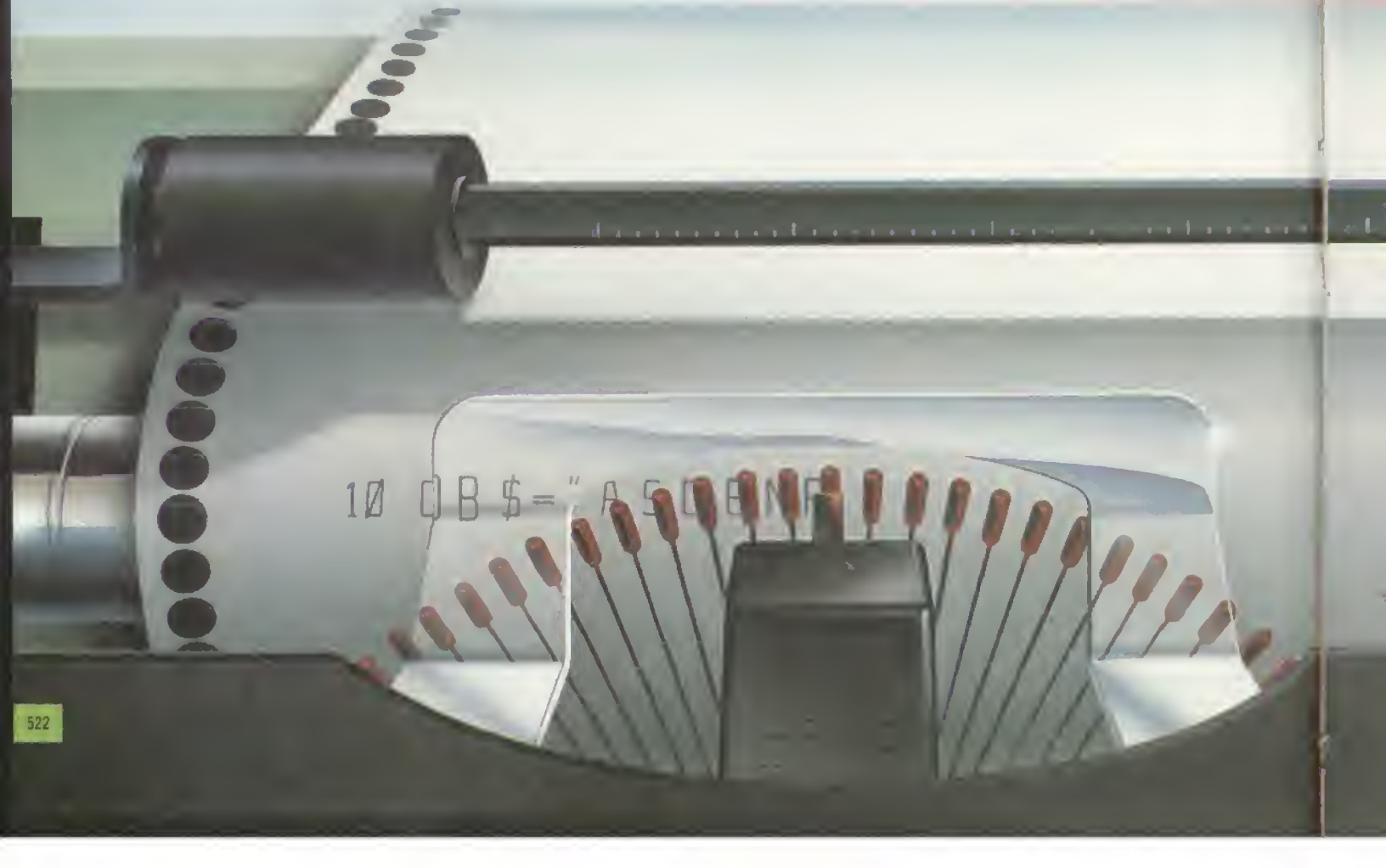
No Brasil, o usuário de computadores pessoais conta com três espécies de impressora: a matricial de impacto, a do tipo "margarida" e a impressora térmica. Um quarto modelo é a impressora-traçadora, muito utilizada na Europa e no Japão; essa espécie, porém, é encontrada no Brasil apenas como parte do equipamento do microcomputador de bolso PC-1500, da Sharp.

Como acontece com todas as impressoras modernas, a matricial conta com carro de papel fixo. A cabeça de impressão se movimenta no sentido transversal sobre o papel; ela contém uma coluna vertical de agulhas bem próximas umas das outras e ligadas a indutores do tipo solenóide. Quando um solenóide é ativado, a agulha é propulsionada contra uma fita tintada, imprimindo um ponto no papel. Os caracteres são formados por meio da combinação de agulhas, e da movimentação da cabeça de impressão.

As primeiras impressoras matriciais tinham matrizes de 5 x 7 (cinco colunas por sete fileiras), porém as mais modernas já utilizam nove agulhas, para produzir um padrão de 7 x 9 ou 9 x 9. Este é um aspecto importante, pois a qualidade final dos caracteres impressos depende do número de pontos na matriz. Uma das características "negativas" das impressoras matriciais, quando operam com matrizes menores, é a impossibilidade de traçar a parte da letra que fica abaixo da linha de impressão, como nas letras p e q.

Essas impressoras são as mais rápidas de todas, existindo modelos com velocidades que variam de 80 a 400 cps (caracteres por segundo). A velocidade máxima, no entanto, raramente é atingida. As impressoras mais modernas imprimem linhas de texto nas duas direções (impressão bidirecional).

Mesmo nos melhores modelos de impressoras matriciais, os caracteres impressos, embora claramente legíveis, são nitidamente formados por pontinhos. Entretanto, é possível conseguir uma qualidade melhor de apresentação. O recurso mais utilizado para isso é a impressão múltipla, que faz com que a cabeça imprima duas vezes cada linha de texto: uma vez em cada direção. Antes da segunda passagem, o rolo de papel é deslocado em uma fração de milímetro, o que causa um pequeno desalinhamento entre as agulhas de impressão e os caracteres já impressos. Assim, os espaços em branco deixados na primeira passagem são preenchidos na segunda, e os caracteres ficam com um aspecto final mais completo e mais escuro (qualidade carta).



Esse recurso, porém, reduz a velocidade normal de impressão à metade e tem efeito apenas em impressoras matriciais com capacidade gráfica. Nestas, o computador controla não só o disparo individual de cada agulha, como o grau de deslocamento horizontal da cabeça de impressão e vertical do rolo.

Existe ainda um tipo de cabeça com um conjunto duplo de agulhas. Esse dispositivo não diminui a velocidade de impressão, mas é menos usado por ser mais caro. Com ele, os caracteres são formados em passos múltiplos: a primeira coluna de agulhas dispara, em seguida vem a segunda, com um ligeiro deslocamento, e assim por diante.

As impressoras gráficas permitem também a troca dos tipos de letra (conhecidos como fontes); desse modo, pode-se controlar a largura e altura dos caracteres, e determinar se eles serão impressos em negrito, ou em itálico, etc. (alguns programas têm comandos internos que indicam o tipo de letra a ser utilizado na impressão).

Embora todos esses métodos representem um progresso considerável, a qualidade final de impressão de textos não é irrepreensível. Para aperfeiçoá-la, é necessário utilizar as impressoras de caracteres formados, ou seja, as que têm caracteres em relevo, como as máquinas de escrever (dos tipos "esfera", "vareta", ou "margarida").

IMPRESSORAS DE CARACTERES FORMADOS

As primeiras impressoras de caracteres formados acopladas a micros foram adaptadas de máquinas de escrever elétricas, do tipo "esfera". Embora a qualidade de impressão seja boa, a massa da cabeça de impressão (que não foi feita para as velocidades máximas de 15 cps que a impressora consegue) torna a impressão lenta e pouco confiável. Conseqüentemente, hoje dominam as impressoras do tipo "margarida".

As impressoras "margarida" empregam um elemento de impressão removível semelhante a uma flor, com "pétalas" irradiando do anel central. Cada "pétala" tem na ponta um caractere em relevo. Para imprimir, a impressora gira a margarida até que a "pétala" com o caractere fique em posição; um pequeno martelo pressiona-a então contra a fita de impressão e o papel. A qualidade de impressão é igual ou superior à de uma máquina de escrever elétrica, e muito superior à de uma impressora matricial (em qualidade carta); entretanto, a velocidade de impressão é bem menor, variando entre 8 e 80 cps.

Tais máquinas contam com recursos de ênfase de texto, como sublinhamen-

Dispostas em torno de um anel (acima), as agulhas da impressora "margarida" são equipadas com caracteres em relevo que imprimem sobre o papel (abaixo).

to, negrito etc. Além disso, elas possibilitam variações no espaçamento entre os caracteres. Algumas são capazes de gerar gráficos por meio do controle da movimentação do papel.

IMPRESSORAS TÉRMICAS

Uma desvantagem importante das impressoras matriciais e do tipo "margarida" é o custo relativamente alto. Assim, tem-se tentado, ao longo dos anos, utilizar outras técnicas mais baratas de impressão.

As primeiras tentativas levaram ao surgimento das impressoras térmicas, que exigem um papel especial, revestido com uma substância química termossensível. Esse papel é caro, mas tem a vantagem de não precisar de fita de impressão. A cabeça impressora é constituída de uma matriz equipada com pequenas peças de aquecimento. O computador seleciona o caractere a ser impresso, e o sistema eletrônico interno da impressora faz passar uma corrente elétrica nas agulhas da cabeça, aquecendo-as até cerca de 150°C. Esse padrão é então aplicado sobre o papel termossensível, escurecendo-o nos pontos correspondentes.

Tal gênero de impressora não pode ser usado para imprimir textos de melhor qualidade. O papel, geralmente de rolo, é alimentado por fricção (rolo de borracha). Assim, seu deslocamento não tem a precisão oferecida pela alimentação por trator (formulário perfurado nas margens). Entretanto, ele apresenta vantagens como: pequeno tamanho, menor desgaste mecânico etc.

Outro tipo é a chamada impressora-traçadora, cujo funcionamento se baseia no mesmo princípio do traçador digital de desenhos. Além de impressora, ela é também traçadora, utilizando pequenas canetas esferográficas de várias cores. A cabeça de impressão típica contém quatro penas, que são alojadas em um tambor rotatório. Sob comando de software, a cabeça pode ser rodada para colocar em posição a caneta com a cor adequada. Ademais, existe um solenóide que retira ou pressiona a pena contra o papel. A cabeça se movimenta no sentido transversal, e o papel no vertical, possibilitando traços contínuos sobre o papel. A velocidade de impressão se situa em torno de 12 cps. O papel é de rolo, semelhante ao das máquinas de calcular.

O PAPEL PARA IMPRESSORA

Ao selecionar uma impressora, é necessário saber antes como se quer a impressão. Isso se aplica não somente aos modelos e à qualidade dos caracteres, como também ao tipo, tamanho, cor e textura do papel. Além disso, a forma com que este é alimentado na impressora também é importante. Por exemplo, se um médico precisar imprimir receitas em papel timbrado, deverá dispor de uma impressora capaz de ser alimentada com folhas soltas.

Existe uma grande variedade de papéis para impressora de largura padronizada. Esta pode ser especificada em número de colunas (80, 120 e 132 colunas) ou ainda em centímetros (os tamanhos variam, neste caso, entre 10 e 40 cm). Os comprimentos são igualmente padronizados (ofício, A4 etc.).

Algumas impressoras usam apenas um tipo de papel (por exemplo, formulário contínuo de oitenta colunas); outras comportam vários tipos. O papel pode ser comprado em folhas soltas, em rolo ou em formulário contínuo. Este último é o papel tradicionalmente utilizado em computadores: consiste de uma faixa contínua dobrada em "Z", e acondicionada em caixas. As folhas podem ser separadas na dobra, por meio de um serrilhado ou corte vincado. Nas bor-

Além de gráficos e desenhos de todos os tipos, a impressora-traçadora pode produzir textos com caracteres de diversas cores e tamanhos.

das, existem duas fitas de papel perfurado (chamadas remalinas), que também podem ser separadas manualmente.

Cada tipo de papel exige uma forma peculiar de alimentação. Os principais mecanismos de alimentação são: trator (rodas dentadas que movimentam o formulário contínuo) e fricção (rolo de borracha, semelhante ao da máquina de escrever, usado com folhas soltas ou com papel de rolo). As folhas soltas podem ser alimentadas manualmente ou por intermédio de um dispositivo especial (este deve ser comprado separadamente, e não existe para todos os tipos de impressora).

Os papéis podem ser lisos (em uma só cor, geralmente branca), pautados e zebrados (com faixas de cor e brancas, alternadas). Existem ainda papéis com cópias múltiplas (1 a 7), com ou sem papel carbono intercalado.

INTERFACES

Por outro lado, o computador só pode se comunicar efetivamente com a impressora se for usada uma interface adequada.

Uma interface é, essencialmente, o hardware que torna possível a conexão entre dois sistemas. Ela é equipada com um circuito eletrônico e com o software necessário para operá-lo. Muitos computadores já são vendidos com a interface para impressora embutida; assim, basta adquirir um cabo para efetuar a ligação. Outros são normalmente vendidos sem qualquer tipo de interface para impressoras.

Quando o computador e a impressora são do mesmo fabricante, há *geralmente* compatibilidade entre eles. Caso isso não aconteça, eles podem ser transformados por meio de uma interface.

É importante também assegurar que o software que se pretende utilizar seja compatível com a impressora, principalmente se ele precisar de recursos especiais (elaboração de gráficos, processamento de textos etc.), peculiares a determinados tipos de impressora. Alguns softwares têm um módulo de instalação que permite selecionar a impressora a partir de um menu.

Para assegurar algum grau de compatibilidade entre os equipamentos no mercado, foram desenvolvidos diversos padrões interfaces. Os dois mais famosos são o RS-232C (uma interface do tipo serial), e o padrão Centronics (uma interface do tipo paralelo).

Numa interface serial, a transmissão dos bits do código de cada caractere é feita gradualmente (um de cada vez). Internamente, porém, tanto o computador quanto a impressora armazenam os bits em paralelo; assim, torna-se necessária a presença de um mecanismo de conversão de paralelo para serial (e vice-versa) em cada ponta do sistema. Daí a vantagem da interface paralela, que manda todos os bits de cada caractere ao mesmo tempo. Apesar de mais rápida, essa interface tem a desvantagem de exigir um cabo com maior número de fios (geralmente coaxial ou paralelo), enquanto a interface serial requer apenas um par de fios.

A conversão, transmissão e recepção de bits em série são efetuadas por um circuito integrado especial chamado UART (*Universal Assynchronous Transmitter and Receiver*, ou Transmissor e Receptor Assincrônico Universal).

Um UART no computador recebe códigos ASCII da memória por meio da UCP, "traduzindo-os" para o formato serial. Além disso, adiciona outros códigos de transmissão e os transmite para a impressora. Um UART idêntico na impressora recebe esses códigos e, se não houver erros, os converte de volta no formato paralelo. As taxas de transferência de dados (medidos em bauds) variam, segundo a impressora, entre 75 e 19 200 bits por segundo (ou bauds), o que equivale a cerca de 7 e 1 800 cps, respectivamente.

Para superar as limitações de velocidade impostas ao computador, devido à lentidão do mecanismo impressor, a maioria das impressoras dispõe de uma memória intermediária chamada *buffer*. A informação a ser impressa é enviada diretamente para o buffer, liberando o computador para outras tarefas. O controlador da impressora esvazia então o buffer na impressão. O tamanho dessa memória intermediária varia entre 8 e 8 000 bytes. À medida que ela aumenta, torna-se menor a freqüência com que o computador é interrompido, ou seja, diminui a demanda da impressora sobre o computador. Assim, embora saia mais caro, é preferível ter uma impressora com um buffer maior. Algumas empresas vendem buffers especiais de grande capacidade para serem instalados entre o computador e a impressora.

Um outro tipo de interface serial empregado durante muito tempo com impressoras é o laço de corrente de 20 mA. Originalmente usado em terminais de teletipos e telex, esse tipo tem sido substituído ultimamente pelo padrão RS-232C, pois é muito lento.

A interface paralela mais usada é a do tipo Centronics. Desenvolvida pelo fabricante do mesmo nome na década de 70, ela foi universalmente adotada devido à sua simplicidade. A esmagadora maioria de impressoras e micros utiliza hoje esse padrão, o que garante ampla compatibilidade.

Outra interface bastante usada, principalmente na interligação de instrumentos de medida a computadores, é o padrão IEE 488, um sofisticado equipamento paralelo que permite a operação full-duplex (comunicação simultânea nos dois sentidos). Como pode ser conectada a muitos micros, essa interface foi incorporada a alguns modelos de impressora. Muitos de seus inúmeros recursos, porém, não são necessários à comunicação computador-impressora. Por esse motivo, a Centronics continua sendo a opção preferida.

CONJUNTOS DE BLOCOS GRÁFICOS (1)

Monstros e espaçonaves são apenas duas aplicações dos blocos gráficos. Extremamente úteis e versáteis, estes podem assumir a forma que quisermos. A imaginação é o limite.

Nem só de monstros e dragões vivem os blocos gráficos. Com eles podemos criar, por exemplo, conjuntos de símbolos especiais ou de novas letras, ou qualquer outra coisa que desejarmos. Neste artigo, veremos como criar uma grande variedade de blocos e como utilizar melhor o programa gerador apresentado nas lições anteriores.

QUAIS SÃO OS LIMITES?

Alguns microcomputadores limitam o número de blocos que podem ser produzidos. Este é o caso do Spectrum, que só permite a criação de 21 blocos. Já no MSX é possível produzir 256 blocos para cada terço da tela de alta resolução — 256 também é o máximo que o programa gerador cria de cada vez. No Apple, o número de blocos é limitado apenas pela quantidade de memória disponível; esse é também o limite do TRS-Color. Neste último, aliás, não se pode falar propriamente de blocos gráficos, mas sim de matrizes que armazenam o padrão desejado. O ZX-81 e o TRS-80 não dispõem desse recurso, impossibilitando a criação de blocos gráficos em BASIC apenas.

No caso do Spectrum, há formas de se extrapolar o limite dos 21 blocos, quando se deseja um número maior. Assim, podemos criar uma tela que use muitos blocos — por exemplo, um jogo em que um sapo deve atravessar uma pista movimentada, pela qual trafegam carros, caminhões e ônibus. Isso certamente consumirá mais de 21 caracteres, sem contar os que vamos precisar para desenhos de fundo como margens de rios, calçadas e acostamentos.

Suponhamos que se trate de um desenho representando uma rua. Teremos, nesse caso, de utilizar blocos para figurar os edifícios, as pessoas, os veículos e outros detalhes da cena. No próximo artigo, explicaremos como criar uma tela desse tipo.

Finalmente, podemos precisar de um novo conjunto de caracteres — o alfabeto grego, por exemplo — o que ultrapassaria o limite de 21 caracteres.

Em resumo, há inúmeras situações que exigem mais de 21 blocos gráficos.

- QUANTOS UDG PODEMOS CRIAR?
- RESERVE UM ESPAÇO NA MEMÓRIA PARA OS BLOCOS
- ESCREVA NA TELA DE ALTA RESOLUÇÃO DO APPLE
- UM NOVO CONJUNTO DE CARACTERES NO SPECTRUM E NO TK-2000
- A ORGANIZAÇÃO DA MEMÓRIA DE ALTA RESOLUÇÃO DO MSX

Felizmente, casos como esses não ficam sem solução, pois existe uma maneira de aumentar o número de blocos oferecidos pelo computador.

O espaço reservado pelo Spectrum ao ser ligado corresponde a 21 blocos gráficos. Podemos, contudo, aumentá-lo, destinando uma área maior na RAM especialmente para esse fim. Assim, quando os padrões dos novos blocos estiverem na memória, serão formados vários "bancos" de 21 blocos cada um.

COMO COLOCAR OS BLOCOS NA MEMÓRIA

A primeira coisa a fazer é encontrar um lugar seguro para guardar nossos blocos na memória, de tal forma que eles não possam ser apagados pelo BASIC. Devemos também decidir quantos blocos usaremos para dimensionar previamente o espaço a ser reservado.

Se quisermos criar mais 21 blocos, podemos fazer o cálculo do comprimento que o "banco de memória" vai ocupar, multiplicando 21 (o número de novos blocos) por 8, que é o número de bytes necessários para definir o padrão de um bloco. O resultado dessa multiplicação é o número de bytes que precisamos reservar.

O passo seguinte consiste em escolher a posição em que vamos colocar o "banco". Quanto mais alto o colocarmos na memória, mais espaço sobrará para o programa em BASIC. O ideal seria posicionar nosso "banco" imediatamente abaixo da área em que ficam guardados os 21 primeiros blocos e reservada automaticamente pelo micro.

O endereço máximo até onde o BASIC pode chegar é denominado RAMTOP. Ele fica situado uma posição abaixo do primeiro byte da área dos blocos gráficos — UDG —, reservada automaticamente pelo Spectrum. Veremos mais tarde que esse endereço não é fixo. Podemos verificar o valor atual de RAMTOP por intermédio do comando:

```
PRINT USR "A" - 1
```

Essa linha mostra na tela o endereço do primeiro byte do bloco definido pelo usuário no lugar "A" menos 1. Se não houver alterações depois que o aparelho for ligado, o valor de RAMTOP deverá ser 65367 no Spectrum de 48K, e 32599 no de 16K.

Agora que sabemos onde fica o RAMTOP, podemos calcular o endereço inicial de nosso espaço reservado para os novos blocos.

Para isso, temos que calcular quantos bytes vamos reservar a partir do primeiro byte da área dos UDG. Nosso banco tem um comprimento de 21 x 8, ou 168 bytes. Dessa forma, ele deve começar 168 bytes abaixo da área dos UDG. Isso fica na posição:

```
PRINT USR "A" - 169
```

que corresponde ao valor 65199, ou 32431 nas máquinas de 16K.

PROTEÇÃO DO BANCO DE BLOCOS

Entretanto, não basta colocar simplesmente os padrões dos blocos nessa área. Programas BASIC muito longos podem chegar até lá, destruindo os dados. Para proteger essa área, devemos

digitar, ou colocar em uma linha do programa, a instrução:

```
CLEAR USR "A" - 169
```

Isso fará com que o RAMTOP seja deslocado 168 bytes para baixo, de modo que as posições acima de 65199 (ou 32431) ficarão protegidas.

Agora, temos que informar o Spectrum onde encontrar os dados; depois disso, colocamos na memória os bytes correspondentes aos novos caracteres.

COMO USAR APONTADORES

Para informar ao computador onde se encontram os dados referentes aos novos blocos, precisamos modificar o valor de um *apontador*.

Existem vários apontadores na memória do Spectrum: cada um deles "aponta" para uma posição importante da memória. Neste caso específico, estamos interessados naquele que "aponta" para o endereço do primeiro byte da área de blocos gráficos. Antes de usar um bloco gráfico, o computador deve descobrir onde seus bytes estão guardados. Para isso, ele precisa primeiro obter o valor do endereço que está no apontador. Esse valor é geralmente 65368, mas devemos modificá-lo, de maneira que ele passe a "apontar" para o início do novo banco de blocos.

Uma característica dos apontadores é que podemos colocar novos valores dentro deles usando **POKE**; assim, o computador utilizará simplesmente o novo endereço. É essa característica que faz do apontador um elemento tão útil na expansão do limite de blocos. Na verdade, precisamos de dois comandos **POKE** para modificar o conteúdo de um apontador, pois o valor máximo que um byte pode conter é 255, ao passo que os valores de endereços são geralmente maiores. Desse modo, o Spectrum quebra o endereço em duas partes, equivalentes, por assim dizer, ao algarismo das dezenas e ao algarismo das unidades de um número normal.

Para calcular as diferentes partes de um número decimal, devemos dividi-lo por 10, caso ele tenha dois algarismos. Tomemos o número 56 como exemplo: para encontrar o algarismo das dezenas, dividimos 56 por 10 e obtemos 5. O resto da divisão, 6 no nosso exemplo, é o algarismo das unidades.

Quando quisermos colocar um novo valor dentro do apontador (usando **POKE**), devemos quebrar o número em duas partes de maneira similar. A diferença é que, em vez de dividirmos por 10, dividimos por 256. Nosso banco contendo 21 novos blocos deve começar no endereço 65200 (no Spectrum de 48K). Assim, é preciso colocar esse valor dentro do apontador, quebrando-o em duas partes. Dividimos então 65200 por 256. Obtemos 254 e um resto igual a 176.

O sistema operacional do Spectrum interpreta a primeira parte de um número com duas partes como sendo a menor (ao contrário do que fazemos com números decimais). Desse modo, o quociente da divisão vai para a segunda parte do número e o resto para a primeira.

O nosso apontador corresponde então a dois endereços: 23675 e 23676. Portanto, precisamos de dois **POKE**:

```
POKE 23675,176
POKE 23676,254
```

Falta agora criar e colocar os novos blocos na memória.

Uma vez modificado o conteúdo do apontador, procedemos exatamente da mesma maneira que com blocos normais: **POKE USR "A"** com o primeiro byte do primeiro caractere, **USR "A" + 1** com o segundo byte do primeiro caractere, e assim por diante.

Se você preferir, pode colocar os valores na memória antes de mudar o apontador. Neste caso, terá que usar o valor do endereço, e não mais **USR "A"**. Assim, no nosso exemplo, o primeiro endereço seria 65200. Para não confundirmos os números, é aconselhável usar **+ 1** em cada novo **POKE**.

Para trabalhar com os caracteres novos, é preciso modificar o apontador (geralmente, é mais fácil fazer isso no início).

Se seguirmos corretamente as instruções, será fácil imprimir os blocos, usando um programa BASIC. Contudo, quando precisamos de novos caracteres ou blocos disponíveis via teclado, o método não é muito adequado. Não é agradável mudar o apontador ou usar o modo gráfico a todo instante.

UM NOVO CONJUNTO DE CARACTERES

Para contornar esse problema, é necessário que redefinamos o conjunto de

caracteres do Spectrum. Isso vai fazer com que o ato de pressionar uma tecla imprima um novo caractere ou bloco gráfico, em vez da letra ou número correspondentes.

Há muitos motivos para se criar um novo conjunto de caracteres: por exemplo, quando queremos imprimir mensagens na tela em outra língua (russo ou grego, digamos), ou quando desejamos personalizar um programa, usando tipos que nós mesmos criamos.

A redefinição do conjunto de caracteres do Spectrum é na realidade bem similar à criação de novos bancos de UDG: reserva-se uma área na memória RAM e coloca-se ali os padrões de maneira análoga.

O único problema com a redefinição é que, mesmo que queiramos criar apenas um caractere novo, todos os outros devem ter seus padrões transferidos para a mesma área da RAM.

A razão para isso é que devemos modificar o apontador do conjunto de caracteres, da mesma maneira que anteriormente mudamos o apontador de UDG. Uma vez modificado o valor do apontador, o computador passa a procurar os padrões das letras no novo endereço na RAM, não podendo retornar à ROM para obter os padrões das letras que não foram modificadas.

Um procedimento útil é transferir para a área reservada na RAM todos os bytes das letras antes de colocar os dados correspondentes aos caracteres que queremos modificar.

Digite a instrução **RANDOM O** para limpar a memória e depois carregue o seguinte programa:

```
 10 CLEAR USR "A"-769
 20 LET d=PEEK 23730+256*PEEK
 23731+1
 30 FOR n=15616 TO 15616+767
 40 POKE d,PEEK n
 50 LET d=d+1: NEXT n
 60 POKE 23606,PEEK 23675
 70 POKE 23607,PEEK 23676-4
 80 LET p=PEEK 23606+256*PEEK
 23607+48*8
 90 FOR n=p TO p+79: READ a:
 POKE n,a: NEXT n
110 DATA 0,124,76,84,86,102,
126,0
130 DATA 0,8,8,8,24,24,24,0
150 DATA 0,126,2,126,96,96,126
,0
170 DATA 0,124,4,126,6,6,126,0
190 DATA 0,96,98,98,126,2,2,0
210 DATA 0,124,64,126,6,6,126,0,
230 DATA 0,62,32,126,98,98,126
,0
250 DATA 0,124,4,4,6,6,6,0
270 DATA 0,60,36,60,102,102,
126,0
290 DATA 0,124,68,126,6,6,126,
0
```

As oito primeiras linhas protegem uma área na RAM e colocam ali os bytes correspondentes ao conjunto de caracteres obtidos na ROM. As linhas restantes redefinem os números.

Coloque o programa para funcionar, aguardando a mensagem "OK". A seguir, aperte uma das teclas numéricas.

Se pressionarmos **NEW**, os números voltarão à sua forma original. Entretanto, os dados correspondentes continuam na memória e o novo conjunto de caracteres pode ser reativado por:

```
POKE 23606, PEEK 23675
POKE 23607, PEEK 23676 - 4
```

Não importa o tamanho da memória de seu micro — 16K ou 48K —, o programa descobre e altera os endereços relevantes. Ele faz isto verificando o valor do apontador do **RAMTOP** (endereços 23730 e 23731). Os valores dos comandos **PEEK** são combinados de maneira a produzir o valor de **RAMTOP**.

O conjunto de caracteres da ROM é guardado entre os endereços 15616 e 15616 + 767 (pelo menos, esta é a parte do conjunto de caracteres que pode ser redefinida). Usando um laço **FOR...NEXT**, o programa transfere para a RAM os bytes dessa região da ROM.

O laço atualiza o endereço que está sendo transferido automaticamente (já que o contador do laço é também o endereço que está sendo transferido); o endereço da RAM que está recebendo o byte é atualizado na linha 50 (variável **d**). Como podemos ver na linha 30, não há necessidade de somar os endereços nem os bytes, já que o Spectrum pode fazer isso por nós. As linhas 60 e 70 modificam o valor do apontador do conjunto de caracteres, para que ele "aponte" para o endereço inicial da área que reservamos na RAM. Esse endereço é calculado a partir do endereço da área de caracteres UDG, obtido por intermédio de **PEEK** (esse mecanismo funcionará mesmo que tenham sido criados novos bancos de UDG). O novo conjunto de caracteres ficará logo abaixo da área de caracteres UDG; assim, o cálculo na linha 70 simplesmente subtrai o comprimento do conjunto de caracteres do endereço inicial da área de UDG. Esse comprimento é igual a 4, mas, como está no byte mais significativo do endereço, ele é multiplicado por 256, resultando no comprimento real do conjunto, que é de 1024 bytes.

Agora que o conjunto de caracteres foi todo transferido para a RAM, podemos substituir alguns dos velhos caracteres por novos.

Se não tomarmos cuidado, corremos o risco de substituir os caracteres errados. Isso pode ser evitado se escolhermos direito os caracteres que serão modificados e se colocarmos os novos bytes nos lugares certos.

O apêndice I do manual do Spectrum apresenta uma lista com o conjunto de caracteres. Somente podem ser redefinidos os caracteres com códigos ASCII entre 32 e 127 (a coluna da esquerda traz o código de cada caractere).

Para calcular os bytes que devem ser modificados, multiplicamos o código do caractere por 8. Essa operação nos dá o endereço do primeiro byte do caractere que queremos redesenhar. Para o caractere de espaço (código ASCII 32), o resultado é 32 × 8 = 256.

Agora que sabemos onde o byte está, dentro do conjunto de caracteres, somamos esse número ao endereço que se encontra no apontador do conjunto de caracteres. Isso é calculado como **PEEK 23606 + 256*PEEK 23607 + 32*8**. No Spectrum de 48K recém-ligado, o resultado será 64600.

Depois de tudo isso, torna-se fácil fornecer os dados do novo caractere ao Spectrum. Usamos **POKE** para colocar oito bytes na memória a partir do endereço que acabamos de calcular. Esses bytes devem corresponder ao padrão do novo bloco.

As linhas que se seguem redefinem o formato dos caracteres de espaço, fazendo com que eles se tornem visíveis na listagem:

```
10 FOR X=64600 TO 64600+7
20 READ A
30 POKE X,A
40 NEXT X
50 DATA 0,126,66,66,66,66,126
,0
```

Se seu micro tem 16K, mude o número 64600 para 31832. Evidentemente, é uma boa idéia usar laços **FOR...NEXT**

para colocar bytes na memória, uma vez que o laço muda automaticamente o endereço e evita que escrevamos oito comandos **POKE**.

Tente agora calcular o endereço inicial dos bytes correspondentes aos diversos caracteres numéricos: você poderá conferir o resultado examinando a listagem do programa.

Já vimos como mudar o formato dos caracteres do MSX. Agora, vamos procurar esclarecer a organização da memória de alta resolução (**SCREEN 2**) para que possamos criar blocos gráficos (à mão ou por meio do gerador de blocos). O artigo *Os Comandos PEEK e POKE* (página 261) apresenta uma introdução ao assunto, que deve ser revista.

A tela do MSX tem 256 pontos de largura por 192 de altura. Cada um desses pontos pode estar aceso ou apagado, assumindo assim a cor de frente ou a cor de fundo, respectivamente. Existem dezesseis cores disponíveis. No modo de alta resolução (**SCREEN 2**), esses pontos estão distribuídos por 768 blocos gráficos de 8 x 8 pontos. Cada bloco gráfico definido pelo usuário pode ocupar uma dessas 768 posições — ao contrário dos sprites, que podem ocupar qualquer posição.

O padrão de cada uma dessas posições (ou blocos) é definido da maneira usual, ou seja, cada linha de oito pontos corresponde a um número de oito bits ou um byte. Assim, cada posição precisa de oito bytes de memória.

O MSX tem uma memória separada só para o vídeo: a memória VRAM. O comando **SCREEN** determina como essa memória será utilizada pelo micro. No modo de alta resolução — que obtemos logo após acionar o comando **SCREEN 2** dentro de um programa BASIC —, uma porção da VRAM é separada para armazenar os padrões de todas as 768 posições da tela. Essa parte da VRAM é denominada *tabela de padrões* e tem seu endereço inicial (na VRAM) contido em **BASE(12)**, que é uma variável interna do micro. Como cada posição — 8 x 8 pontos — da tela necessita de oito bytes, a tabela de padrões tem um comprimento igual a 768 x 8 = 6144 bytes. Cada byte define o padrão de uma linha de bloco gráfico.

Esse padrão, por sua vez, determina quais pontos estão acesos e quais pontos estão apagados. Para colorir os pontos, o computador precisa usar uma outra área da VRAM, a *tabela de cores*, que tem endereço inicial armazenado em **BASE(11)**. Cada linha de bloco gráfico pode ter apenas uma cor de frente (pontos acesos) e uma de fundo (pontos apagados). Assim, a cada linha de bloco gráfico — ou posição — da tela corresponde um byte de cor. O valor desse byte é igual ao código da cor de fundo mais 16 multiplicado pelo código da cor de frente. Como a tela tem 768 posições e cada posição oito linhas, a tabela de cores também tem 6144 bytes de comprimento.

Quando quisermos colocar um bloco gráfico em determinado ponto da tela, devemos calcular as posições nas tabelas de padrões e de cores correspondentes aos bytes que vamos alterar. Para conhecermos a posição na tabela de cor ou na de padrões do primeiro byte que coloca um bloco na tela, multiplicamos a posição desejada na tela por oito. Se quisermos colocar um bloco, digamos, na posição 25 da tela, temos que alterar oito bytes, a partir da posição 25 x 8 = 200 na tabela de padrões e na de cores. O programa a seguir tenta ilustrar o processo:

```
5 P=367
10 SCREEN 2
20 FOR I=0 TO 7
30 READ A(I)
40 VPOKE BASE(12)+P*8+I,A(I)
50 VPOKE BASE(11)+P*8+I,6*16+11
60 NEXT I
70 GOTO 70
100 DATA 254,124,56,16,8,28,62,
127
```

Este programa desenha na posição 367 da tela o bloco gráfico cujo padrão corresponde aos oito bytes da linha **DATA** 100.

A linha 5 determina a posição P da tela onde será colocado o bloco gráfico. A linha 10 seleciona o modo de alta resolução.

O comando **VPOKE** é utilizado para colocar valores na VRAM. Note como os endereços correspondentes ao padrão e à cor das linhas do bloco são calculados. Como são oito linhas, o laço **FOR...NEXT** será repetido oito vezes. Na linha 40, o endereço do primeiro byte a ser alterado na tabela de padrões é 8 multiplicado pela posição na tela P. Essa posição é contada na tabela a partir de seu endereço inicial na VRAM, dado por **BASE(12)**. A variável **I**, contador do laço, faz com que oito posições consecutivas da VRAM sejam modificadas. Por fim, o valor colocado na tabela de padrões é A(I), que foi lido com **READ** na linha 30. A cor de cada linha do bloco é determinada de forma semelhante. O cálculo da posição é o mesmo, só que o valor obtido é contado a partir de **BASE(11)**, endereço inicial da tabela de cores na VRAM. O byte de cor determina que a cor de frente seja vermelha (código 6) e a de fundo seja amarela (código 11), na linha 50.

A linha 70, por sua vez, evita que a tela seja apagada, já que o modo gráfico de alta resolução — **SCREEN 2** — só permanece ativo enquanto o programa estiver funcionando.

O PROGRAMA GERADOR DE BLOCOS

Se você já criou vários blocos gráficos com o auxílio de nosso programa gerador, chegou o momento de usá-los. Se você ainda não usou o programa, o que vem a seguir pode parecer confuso. Assim, para aproveitar bem as próximas linhas, é necessário que você tenha criado e guardado em fita todo um banco cheio de blocos gráficos criados pelo gerador.

Uma vez gravado um banco contendo 256 blocos, podemos recuperá-lo de duas maneiras: usando o próprio programa gerador para fazer uma edição, por exemplo, ou para retirá-lo diretamente da memória do micro e usar os blocos em outro programa. Na segunda hipótese, a primeira coisa a fazer é proteger uma área no topo da memória do micro, colocando ali o banco de blocos. Para isso, digite:

```
CLEAR 200,&HC999
```

Em seguida, posicione a fita e carregue o banco de blocos usando:

```
BLOAD "CAS:"
```

e o banco terá sido carregado.

Para trazer o banco para a tela, use um programa BASIC:

```
5 SCREEN 2
10 FOR J=0 TO 2*256*8 STEP 256*
8
20 FOR I=0 TO 256*8-1
```

```
30 VPOKE BASE(12)+I+J,PEEK(&HD1
00+I)
40 VPOKE BASE(11)+I+J,PEEK(&HE1
00+I)
50 NEXT I,J
60 GOTO 60
```

Depois de executar o programa, teremos três cópias do banco de blocos que criamos anteriormente. Todos os blocos produzidos estarão lá. Se não tivermos armado um banco completo com 256 blocos, alguns blocos estranhos apareceram, mas sem prejuízo dos que foram criados. Esses blocos exteriores correspondem a valores da memória que permaneceram inalterados quando criamos o banco e foram gravados junto com ele. No entanto, nada disso acontecerá se criarmos 256 blocos, preenchendo inteiramente o espaço.

A linha 5 ativa a tela de alta resolução. A linha 10 se prepara para fazer três cópias do banco, repetindo três vezes as linhas de 20 a 40. O laço entre as linhas 30 e 40 será repetido 256 x 8 vezes — correspondente a 256 blocos com oito bytes cada um.

A linha 30 transfere os padrões do banco — endereço inicial D100, em hexa — para a tabela de padrões — endereço inicial **BASE(12)**. A linha 40 transfere os bytes de cor do banco — endereço inicial E100, em hexa — para a tabela de cores — endereço inicial **BASE(11)**. Observe como esse processo é demorado.

Como vemos, o banco de blocos criado pelo gerador consiste em uma "tabela de padrões" — endereço inicial D100, hexadecimal — e uma "tabela de cores" — endereço inicial E100.

Obviamente, nem sempre desejamos transferir de uma só vez todo o banco para a tela, como no exemplo. Quando quisermos obter o padrão (ou a cor) de um bloco isolado que está no banco, teremos que calcular onde se encontram os bytes correspondentes. Para isso, precisamos saber qual a posição desse bloco no banco (é necessário fazer uma lista quando criarmos os blocos). Conhecendo esse valor, é possível calcular o endereço do primeiro byte do bloco, multiplicando sua posição por 8 e somando o resultado ao endereço inicial — D100 ou E100, conforme queiramos padrões ou cores.

A TABELA DE NOMES

Por esse método de trabalho com a tela de alta resolução, tudo o que foi colocado nas tabelas de padrões e cores apareceu imediatamente na tela. Existe, porém, outra maneira de usar essas tabelas: levando-as a funcionar como bancos de blocos gráficos, dos quais escolhemos apenas alguns para exibir de cada vez na tela.

Para fazer isso, utilizamos a tabela de nomes — endereço inicial BASE (10), na VRAM. Essa tabela tem 768 bytes de comprimento — cada byte corresponde a uma posição (8 x 8 pontos) da tela. Ela permite que codifiquemos os blocos que estão na tabela de padrões com números de 0 a 255, como no banco do gerador. Para que o bloco de número 37, por exemplo, apareça na posição 170 da tela, basta fazer com que o byte 170 da tabela de nomes seja igual a 37, usando **VPOKE BASE(10) + 170,37**. Essa organização é semelhante à da tela de textos, onde os padrões dos caracteres ficam numa tabela de padrões e uma tabela de nomes faz com que os caracteres sejam impressos na tela codificados por seu código ASCII.

Existem aqui dois problemas. Primeiro, só podem ser codificados 256 blocos gráficos, enquanto o número de blocos nas tabelas de cor e padrão é três vezes maior. Esse problema é contornado com a criação de três bancos de blocos gráficos. Isso significa que os 256 primeiros blocos (256 x 8 bytes) das tabelas de cor e padrão têm seus códigos válidos apenas para o terço superior da tela. Do mesmo modo, o terço médio e o inferior contam com seus próprios bancos — respectivamente, no terço médio e inferior das tabelas de cor e padrão.

O segundo problema é que nem o próprio micro emprega esse tipo de organização. Assim, se trabalharmos com a codificação descrita para criar telas gráficas, não poderemos recorrer a comandos como **DRAW**, **LINE** e **PAINT**. A forma de organização utilizada pelo computador é a mesma que faz com que cada byte colocado na tabela de padrões e de cores apareça na tela, possibilitando o modo de utilização que descrevemos linhas atrás.

Para entender melhor, veja os valores que o micro mantém normalmente na tabela de nomes:

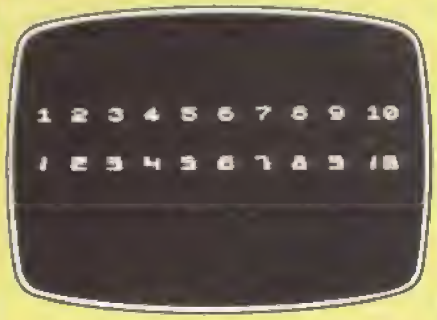
```
20 FOR I=0 TO 3*256-1
30 PRINT I,VPEEK(BASE(10)+I)
40 NEXT
```

Este programa lista os valores contidos na tabela de nomes. Note que a posição 1 corresponde ao caractere 1, que é o primeiro da tabela de padrões e de cores; a posição 2 tem valor 2, correspondente ao segundo bloco da tabela de padrões (e de cores), e assim por diante.

Para familiarizar-se com a utilização da tabela de nomes, acrescente as próximas linhas ao programa que carrega o banco criado pelo gerador. Ele só funcionará, contudo, se houver um banco de blocos no topo da memória.

```
60 FOR I=0 TO 255
70 FOR J=0 TO 3*256-1
80 VPOKE BASE(10)+J.I
90 NEXT J,I
100 GOTO 100
```

Este programa preenche toda a tabela



A figura mostra na linha superior os números empregados pelo Spectrum; logo abaixo, aparecem alinhados os novos números criados pelo programa.

de nomes com o mesmo valor. O código de cada bloco é modificado pelo laço **FOR...NEXT** das linhas 60 a 90, de tal forma que, ao ser desenhado, cada um dos 256 blocos do banco ocupa a tela inteira.

Mais interessante ainda é fazer com que os blocos que estão na tela se movimentem. Para tanto, você deve modificar a linha 80:

```
80 VPOKE BASE(10)+J,(VPEEK(BASE
(10)+J)+1)AND 255
```

O programa movimenta todos os blocos da tela mudando apenas os valores na tabela de nomes. Uma vez copiado o banco do alto da memória nas tabelas de cor e padrão, seus valores não se modificam. A velocidade com que são movimentados os blocos é aproximadamente dezesseis vezes maior que aquela com que eles são desenhados no início do programa, já que um **VPOKE** na tabela de nomes equivale a oito **VPOKE** na tabela de padrões e oito **VPOKE** na tabela de cor.

Esta última modificação faz com que apenas o terço superior da tela seja movimentado:

```
70 FOR J=0 TO 255
```

O maior problema para a utilização de blocos gráficos no Apple e no TK-2000 é a caótica organização da memória de vídeo desses computadores. Com efeito, na tela desses micros, as linhas consecutivas não têm números consecutivos e é necessário recorrer à matemática para contornar a dificuldade, o que torna parte do programa muito complicada para alguns.

Nada melhor do que um exemplo em BASIC para ilustrar o problema:

```
10  HGR
20  FOR I = 8192 TO 16383
30  POKE I,255
40  NEXT I
```

Este programa coloca na tela pedaços de linha, por intermédio de **POKE**. Embora os endereços da memória de vídeo sejam consecutivos, as linhas produzidas não o são. Ora, nossos blocos gráficos só serão desenhados se colocarmos valores na memória de vídeo usando **POKE**.

O problema pode ser resolvido pela dedução de uma fórmula matemática que permita calcular os oito endereços necessários para desenhar um bloco gráfico em uma determinada posição.

ESCREVA NA TELA DE ALTA RESOLUÇÃO

Para os usuários do TK-2000 o programa a seguir pode parecer desnecessário, pois esse micro é capaz de escrever textos na tela de alta resolução. Entretanto, os usuários do Apple o acharão bem útil.

```
10  HGR :E = 16384:T = 8192
20  FOR I = E TO E + 9 * 8 - 1:
 READ B: POKE I,B: NEXT
30  DATA  28,34,38,42,50,34,28,
0
40  DATA  16,24,20,16,16,16,56,
0
50  DATA  28,34,32,24,4,2,62,0
60  DATA  28,34,32,24,32,34,28,
0
70  DATA  16,24,20,18,62,16,16,
0
80  DATA  62,2,2,30,32,34,28,0
90  DATA  28,34,2,30,34,34,28,0
100  DATA  62,32,32,16,8,8,8,0
110  DATA  28,34,34,28,34,34,28
,0
120  DATA  28,34,34,60,32,34,28
,0
130  FOR Y = 0 TO 19: FOR X = 0
 TO 39:N =  INT ( RND (1) * 9)
140  FOR I = 0 TO 7
150  POKE T + (Y - 8 * (Y > 7)
 - 8 * (Y > 15)) * 128 + 40 * (Y
 > 7) + 40 * (Y > 15) + X + 102
4 * I, PEEK (E + N * 8 + I)
160  NEXT I,X,Y
```

O programa começa ativando a tela de alta resolução e estabelecendo os valores de E, endereço inicial da página 2, e T, endereço inicial da página 1. Os usuários do TK-2000 devem mudar o valor de T para 40960.

A linha 20 lê os valores contidos nas linhas **DATA**, criando parte de um banco de blocos.

A linha 130 inicia um laço **FOR...NEXT**, onde Y é a linha e X a coluna em que serão impressos os números. N é o número que será impresso, escolhido ao acaso.

A parte mais importante do programa é o **FOR...NEXT** das linhas 140 a 160. Ela é responsável pelos oito comandos **POKE** que colocam o bloco na posição desejada. A fórmula a que nos referimos é a que está na linha 150.

Este programa só escreve números. Ora, nosso objetivo é escrever todo tipo de mensagem na tela gráfica. Para isso, precisamos criar (com o auxílio do gerador) um conjunto inteiro de caracteres. Quando houver um banco de blocos com caracteres na página 2, poderemos utilizar seus bytes para escrever na tela de alta resolução.

Os usuários do TK-2000 devem agora redefinir o conjunto de caracteres, criando seus próprios tipos gráficos.

Para os que não se habituaram ao uso do gerador, o programa a seguir cria um banco de blocos contendo os caracteres de código ASCII entre 32 e 95.

Uma vez executado com sucesso, ele poderá ser apagado e todo o banco será visualizado e editado pelo gerador de blocos. Basta carregá-lo sem desligar o computador.

As letras têm aqui posições correspondentes a seus códigos ASCII.

```
10  HGR :E = 16384:T = 8192
20  FOR I = E +.32 * 8 TO E + 3
2 * 8 + 64 * 8 - 1: READ B: POK
E I,B: NEXT
100  DATA  0,0,0,0,0,0,0,0
110  DATA  8,8,8,8,0,0,8,0
120  DATA  18,18,18,0,0,0,0,0
130  DATA  18,18,63,18,63,18,18
,0
140  DATA  8 ,60 ,10 ,28 ,40 ,3
0 ,8 ,0
150  DATA  38,38,16,8,4,50,50,0
160  DATA  12,18,18,12,82,34,92
,0 .
```

```
170  DATA  24,16,8,0,0,0,0,0
180  DATA  32,16,8,8,8,16,32,0
190  DATA  2,4,8,8,8,4,2,0
200  DATA  0,8,42,28,28,42,8,0
210  DATA  0,8,8,62,8,8,0,0
220  DATA  0,0,0,0,0,24,16,8
230  DATA  0,0,0,62,0,0,0,0
240  DATA  0,0,0,0,0,24,24,0
250  DATA  64,32,16,8,4,2,1,0
260  DATA  28,34,38,42,50,34,28
,0
270  DATA  16,24,20,16,16,16,56
,0
280  DATA  28,34,32,24,4,2,62,0
290  DATA  28,34,32,24,32,34,28
,0
300  DATA  16,24,20,18,62,16,16
,0
310  DATA  62,2,2,30,32,34,28,0
320  DATA  28,34,2,30,34,34,28,
0
330  DATA  62,32,32,16,8,8,8,0
340  DATA  28,34,34,28,34,34,28
,0
350  DATA  28,34,34,60,32,34,28
,0
360  DATA  0,24,24,0,0,24,24,0
370  DATA  0,24,24,0,0,24,16,8
380  DATA  32,16,8,4,8,16,32,0
390  DATA  0,0,0,62,0,62,0,0
400  DATA  4,8,16,32,16,8,4,0
410  DATA  28,34,32,16,8,8,0,8
420  DATA  28,34,58,42,58,2,60,
0
430  DATA  8,20,34,34,62,34,34,
0
440  DATA  30,34,34,30,34,34,30
,0
450  DATA  28,34,2,2,2,34,28,0
460  DATA  30,34,34,34,34,34,30
,0
470  DATA  62,2,2,62,2,2,62,0
480  DATA  62,2,2,30,2,2,2,0
490  DATA  28,34,2,58,34,34,28,
0
500  DATA  34,34,34,62,34,34,34
,0
510  DATA  24,8,8,8,8,8,28,0
520  DATA  56,16,16,16,18,18,28
,0
530  DATA  34,18,10,6,10,18,34,
0
540  DATA  2,2,2,2,2,2,62,0
550  DATA  65,99,85,73,65,65,65
,0
560  DATA  34,38,42,42,50,34,34
,0
570  DATA  28,34,34,34,34,34,28
,0
580  DATA  30,34,34,30,2,2,2,0
590  DATA  28,34,34,34,42,50,60
,64
600  DATA  30,34,34,30,18,34,34
,0
610  DATA  60,2,2,28,32,32,30,0
620  DATA  62,8,8,8,8,8,8,0
630  DATA  34,34,34,34,34,34,28
,0
640  DATA  34,34,34,34,34,20,8,
0
650  DATA  65,65,65,73,85,99,65
,0
660  DATA  34,34,20,8,20,34,34,
0
670  DATA  34,34,34,20,8,8,8,0
680  DATA  62,32,16,8,4,2,62,0
690  DATA  60,4,4,4,4,4,60,0
700  DATA  1,2,4,8,16,32,64,0
710  DATA  30,16,16,16,16,16,30
,0
720  DATA  8,20,34,0,0,0,0,0
730  DATA  0,0,0,0,0,0,62,0
799  HGR
800 A$ = "ESTA MENSAGEM E' UM T
ESTE"
810 C = 7:L = 11: GOSUB 1000
820  END
1000 N = L * 40 + C
1010  FOR J = 1 TO  LEN (A$)
1020 B$ =  MID$ (A$,J,1)
1030 B =  ASC (B$)
1040 L =  INT (N / 40):C = N -
40 * L
1050  FOR I = 0 TO 7
1060  POKE T + (L - 8 * (L > 7)
 - 8 * (L > 15)) * 128 + 40 * (
L > 7) + 40 * (L > 15) + C + 10
24 * I, PEEK (E + B * 8 + I)
1070  NEXT I:N = N + 1: NEXT J
1080  HCOLOR= 6
1090  HPLOT 25,80 TO 250,80 TO
250,105 TO 26,105 TO 26,80
1100  RETURN
```

A linha 10 estabelece as condições iniciais. A linha 20 cuida de transferir para a página 2 de vídeo os padrões dos caracteres contidos nas linhas **DATA**. Note que são 64 caracteres, a partir da posição 32, e cada caractere precisa de oito bytes para ser definido. Com essas informações fica mais fácil entender os números da linha.

Para imprimir uma mensagem, utiliza-se a sub-rotina que começa na linha 1000. Essa sub-rotina imprime na tela gráfica o conteúdo da variável alfanumérica **A$**. Para fazer isso, ela toma letra por letra da variável, usando **MID$** na linha 1020. Como o código ASCII da letra é igual à sua posição no banco, fica fácil para a linha 1060 imprimir a letra correspondente. C e L são respectivamente a linha e a coluna onde a mensagem é impressa. O laço da linha 1010 faz com que todas as letras sejam lidas, uma a uma, até o final de **A$**. **N** é a posição absoluta do "cursor" na tela.

N é usado em lugar de L e C para evitar que a mensagem ultrapasse os limites da tela, "pulando" de linha. Observe que é **N** que é incrementado na linha 1070 e que **L** e **C** têm seus novos valores calculados a partir de **N**, na linha 1040.

As últimas linhas do programa fazem uma pequena moldura colorida para a mensagem, lembrando ao usuário que se trata da tela gráfica.

As letras aparecerão com cores aleatórias — geralmente tons de amarelo. Isto é inevitável devido à maneira com que o Apple codifica as cores.

Para ver e editar os novos caracteres usando o editor, apague o programa com **NEW** e carregue o gerador. Antes de executá-lo, acrescente as próximas linhas, que dão a ele uma nova função, disponível através de "M".

```
195  IF K$ = "M" THEN  GOSUB 21
10: GOTO 50
```

T

O TRS-Color tem dois comandos — **GEI** e **PUT** —, que permitem a criação e o controle de blocos gráficos. Como esse micro guarda os valores correspondentes ao padrão dos blocos dentro de matrizes, o único limite ao número de blocos é o tamanho da memória. Isso significa que podemos criar tantos blocos quantos couberem em 32k.

Ao contrário dos demais computadores, o TRS-Color não permite que mudemos o formato de seus caracteres da ROM — ou seja, os caracteres disponíveis por intermédio do teclado. Portanto, não podemos alterar os tipos com que são escritas as listagens e as mensagens de erro. É possível, porém, produzir blocos gráficos que sejam ao mesmo tempo letras e dar a estas o formato que se quiser.

Um programa para isso deveria ter uma série de comandos **IF INKEY$ = "X" THEN...**, um para cada letra, o que o tornaria muito lento. Se você optar por uma operação dessas para alguns caracteres, empregue o comando **PMODE1** e as linhas **DATA** do programa do Spectrum.

UM COMPACTADOR DE PROGRAMAS

Ao se digitar um programa em BASIC, é conveniente colocar espaços em branco entre os comandos, as variáveis e os dados, para tornar mais fácil a leitura das linhas.

Necessárias para nossa própria orientação, as linhas **REM** são especialmente importantes quando estamos depurando o programa. O problema é que os espaços entre as palavras e as linhas **REM** diminuem a velocidade de execução do programa e ocupam um grande espaço na memória.

Quando o programa está funcionando bem e desejamos obter um máximo de rendimento, temos que apagar todas essas linhas e espaços. Isso, porém, exige muito esforço e, o que é pior, pode introduzir novos erros no programa.

O programa em código de máquina que apresentamos aqui faz todo esse trabalho para você. Uma vez que seu programa esteja funcionando, execute a rotina em código para compactá-lo.

A seguir, apresentamos um compactador de programas para o TRS-Color. Caso você use o monitor, e não o Assembler, o endereço inicial é 30000.

```
10   ORG 30000
20   LDU 25
30   LDD 27
40   PSHS D
50   BRA LONE
60   LTWO LDA ,X+
70   BNE LTHR
80   LDU ,U
90   LONE CLR ICOM,PCR
100   LDD ,U
110   BNE LFOU
120   LDX 27
130   STX 29
140   STX 31
150   PULS D
160   SUBD 27
170   JSR $B4F4
180   RTS
190   LFOU LEAX 4,U
200   BRA LTWO
210   LTHR CMPA #32
220   BNE LFIV
230   TST ICOM,PCR
240   BNE LTWO
250   LDD #1
260   PSHS D,X,U
270   BSR SHIFT
280   PULS D,X,U
290   LEAX -1,X
300   BRA LTWO
310   LFIV CMPA #34
320   BNE LTWE
330   LDB ICOM,PCR
340   EORB #1
350   STB ICOM,PCR
360   BRA LTWO
370   LTWE CMPA #134
380   BNE LSIX
390   LDA -2,X
400   CMPA #255
410   BEQ LTWO
420   LDB ICOM,PCR
430   EORB #2
440   STB ICOM,PCR
450   BRA LTWO
460   LSIX CMPA #130
470   BEQ LSEV
480   CMPA #131
490   BNE LTWO
500   LSEV LDA -2,X
510   CMPA #255
520   BEQ LTWO
530   LDD ,U
540   LEAX -1,X
550   PSHS X,U
560   SUBD ,S++
570   TFR X,Y
580   LDX ,U
590   LEAX -1,X
600   PSHS D,X
610   TFR Y,D
620   SUBD 4,S
630   CMPB #4
640   BNE LNIN
650   PULS D
660   ADDD #4
670   PSHS D
680   LNIN BSR SHIFT
690   PULS D,X,U
700   LBRA LONE
710   ICOM FCB 0
720   SHIFT LDD ,U
730   BEQ SHTWO
740   LDX ,U
750   SUBD 2,S
760   STD ,U
770   TFR X,U
780   BRA SHIFT
790   SHTWO LDD 4,S
800   SUBD 2,S
810   TFR D,U
820   LDX 4,S
830   SHTHR LDA ,X+
840   STA ,U+
850   CMPX 27
860   BLO SHTHR
870   LDD 27
880   SUBD 2,S
890   STD 27
900   RTS
910   END
```

COMO FUNCIONA

A primeira instrução **LDU 25** carrega o registro U com o conteúdo das posições 25 e 26 da memória. Estas contêm o valor de uma variável do sistema que corresponde ao endereço inicial do programa em BASIC. **LDD 27** carrega o registro D com o conteúdo das posi-

Apague as linhas REM e os espaços inúteis, tornando seus programas em BASIC mais rápidos e econômicos. Existe um programa em código para isso.

- LOCALIZE OS COMANDOS REM E OS ESPAÇOS
- COMO USAR SINALIZADORES
- COMO COMPRIMIR UM PROGRAMA NA MEMÓRIA

ções 27 e 28, que contêm igualmente o valor de uma variável do sistema. Desta vez, porém, trata-se de um apontador, que indica o endereço da primeira posição livre da memória, após o fim do programa em BASIC. Seu valor é guardado na pilha por **PSHS D**.

Mais adiante, durante a execução do programa, será calculado o número de bytes economizado pelo processo de compressão. Assim, é preciso que o computador saiba onde o antigo programa em BASIC termina. A instrução **BRA** (**BR***anch* **A***lways*, ou "desvie sempre") transfere a execução para o meio da sub-rotina seguinte, no rótulo **LONE**.

COMO USAR SINALIZADORES

A linha **CLR ICOM, PCR** limpa a área de armazenamento formada pela instrução **FCB0** que se segue ao rótulo **ICOM**. O comando **PCR** (*Programa Counter Relative*) faz com que o endereçamento utilizado possibilite a realocação do programa quando necessário.

FCB (*Form Constant Byte*) reserva um byte para a armazenagem de dados. Qualquer tipo de dado pode ser guardado ali, mas, no nosso caso, armazenaremos um par de indicadores que o programa usará para saber se está manipulando uma variável alfanumérica. Quando se tratar de um cordão, os espaços em branco não devem ser apagados. Não é conveniente também que as palavras que o BASIC escreve na tela apareçam juntas. Os bits da posição de memória rotulada como **ICOM** serão usados como sinalizadores, do mesmo modo que é feito no registro indicador de condições. Os indicadores serão estabelecidos (ou ligados) quando o programa encontrar uma variável alfanumérica, voltando a valer zero quando o programa estiver cuidando de outras expressões. Desse modo, ao fazer a compactação, podemos verificar os bits de **ICOM** para saber como proceder.

Existe uma instrução parecida com **FCB**. É **FDB** (*Form Double Byte*) que reserva dois bytes para colocar dados.

O 0 após **FCB** coloca inicialmente 00 nesse byte (se o 0 for colocado após **FDB**, os dois bytes reservados receberão 00 00). **CLR ICOM, PCR** limpa o conteúdo do byte a cada linha.

COMO ATUALIZAR OS APONTADORES

LDD,U coloca no registro D o conteúdo do endereço contido em U. Em outras palavras, ele posiciona os dois primeiros bytes do programa em D.

Os dois primeiros bytes de um programa em BASIC correspondem ao endereço do início da linha seguinte. Quando esses dois bytes forem 00 00, chegamos ao final do programa. **BNE** (desvio, se diferente de zero) verifica essa condição. Enquanto não se tratar do final do programa, esses dois bytes serão diferentes de 00 00, e o desvio não será realizado. **LDX** coloca o conteúdo das posições de memória 27 e 28 no registro X. Essas posições contêm o endereço do início da "área das variáveis", isto é, o primeiro byte livre depois do programa BASIC.

Esse apontador será atualizado repe-

tidas vezes à medida que o programa for comprimido. **STX 29** e **STX 31** colocam esse endereço nas posições 29 e 30, e 31 e 32. Esses dois apontadores são variáveis do sistema que correspondem ao endereço inicial da tabela de apontadores das matrizes e ao endereço final da "área" do BASIC. As variáveis serão definidas e as matrizes montadas somente quando o BASIC for executado. Assim, aquelas duas variáveis do sistema devem ficar apontando para o endereço do primeiro byte livre após o final do programa em BASIC, enquanto este é compactado.

ESCREVA NA TELA

O valor contido no apontador do final da área BASIC (colocado na pilha quando o programa ainda tinha o tamanho original, no início da rotina em código) é recuperado da pilha. Em seguida, o valor que ficai no apontador após a compressão do programa será subtraído daquele que acaba de voltar da pilha e o resultado, armazenado em D.

JSR $B4F4 transfere o programa para uma sub-rotina da ROM que toma o valor do registro D, converte-o em decimal e o imprime na tela. Como podemos observar, o registro D contém o número de bytes economizados pela compressão do programa. Quando o processo terminar, a economia de espaço será impressa na tela. **RTS** retorna ao BASIC.

COMO PERCORRER A MEMÓRIA

Se o programa compactador não chegar ao final do programa em BASIC, **BNE LFOU** enviará este último para a próxima ocorrência do rótulo **LFOU**. O comando **LEA** significa "carregar o endereço efetivo" (**L**oad **E**fective **A**dress). Neste caso, ele opera sobre o registro X. **LEA X 4,U** toma o endereço em U, soma 4 e coloca o resultado em X. Os dois primeiros bytes de qualquer linha BASIC são o endereço inicial da linha seguinte; os dois bytes imediatamente posteriores contêm o número da linha. Deste modo, a instrução move o microprocessador para o início dos códigos do BASIC. Em seguida, o programa vai para o rótulo **LTWO**.

LDA,X+ coloca o primeiro byte do BASIC no acumulador. O sinal "+" significa que o registro X é incrementado após a execução da instrução. Isso faz com que o valor contido em X corresponda ao byte seguinte do programa.

BNE LTHR desviará para o rótulo **LTHR** se o conteúdo do acumulador não for zero. Um byte igual a 0 (zero) corresponde ao final da linha BASIC. Quando o programa chegar ao fim da linha, o desvio não ocorrerá e o conteúdo da memória cujo endereço está em U será colocado no registro U por **LDU, U**. No início, o conteúdo da variável do sistema que aponta para o começo do BASIC foi colocado em U. No TRS-Color, os dois primeiros bytes de uma linha correspondem ao endereço da linha seguinte. Cada vez que o programa em código chega ao fim de uma linha BASIC, o registro U é atualizado de modo a corresponder ao endereço inicial da próxima linha. A seguir, é realizado o comando de limpeza **CLR**.

CUIDE DOS ESPAÇOS

Se ainda não se trata do final de uma linha BASIC, ocorre desvio e o comando **CMPA #32** verifica se o byte colocado no acumulador corresponde a um espaço. O código ASCII do caractere de espaço é 32. Caso se trate de um espaço, o valor do acumulador será igualado a 0 (zero) pelo comando **CMPA #32**, de forma que a instrução de desvio **BNE LF,V** não será executada.

TST ICOM,PCR verifica o estado da posição de memória rotulada como **ICOM**. Se qualquer um dos bits indicadores em **ICOM** valer 1, é sinal de que se trata de um cordão, de modo que o espaço não deve ser eliminado e **BNE LTWO** voltará ao início do programa. Caso o espaço esteja entre dois comandos BASIC, o processador passará para a próxima instrução: **LDD #1**.

COMO USAR A PILHA

LDD #1 coloca 1 no registro D. **PSHSD,X;U** transporta o conteúdo des-

ses registros para a pilha da máquina. Pode parecer desnecessário pôr o registro D na pilha, já que sabemos que ele contém 1; contudo, a instrução seguinte, **BSR SHIFT**, desvia para a sub-rotina **SHIFT**. Esta desloca o programa aproveitando o espaço deixado pelos caracteres apagados. Ela é usada quando apagamos espaços — apenas um byte — e quando apagamos linhas **REM** — vários bytes. O valor do registro D corresponde ao número de bytes apagados e é usado pela sub-rotina. No caso de uma linha **REM**, esse valor pode chegar a 255 bytes.

Usamos um comando de salto relativo, **BSR**, em vez de um de salto absoluto, **JSR**, de maneira que o programa seja realocável na memória. O comando **JSR** é mais adequado quando usamos sub-rotinas de ROM, que são fixas.

Depois que a rotina na **SHIFT** for executada, os valores de D,X e U serão recuperados por **PULS D,X,U**. Note que D, X e U são especificados na mesma ordem em que estão guardados na pilha.

Não importa a ordem em que definimos os registros. O microprocessador tem uma ordem pré-determinada para guardar e recuperar grupos de registros. A ordem de armazenamento é: byte menos significativo do registro PC; byte mais significativo de PC; bytes de S ou U, na mesma ordem baixo-alto; Y, X, DP, A, B e CC. A ordem de recuperação é a inversa: CC primeiro, PC por último.

Os registros U e S são os apontadores das duas pilhas — U para a pilha do usuário e S para a pilha da máquina. Obviamente, não podemos colocar o valor de um apontador dentro de sua própria pilha, nem podemos recuperar um valor da pilha, colocando-o dentro de seu apontador. Este é o motivo de termos colocado U ou S na ordem anterior. A própria instrução informa de quê pilha se trata. **PSHS** usa a pilha de máquina e **PSHU** e **PULU**, a pilha do usuário.

LEAX −1,X decrementa o registro X, subtraindo 1 de seu conteúdo. Isso é feito porque o programa acaba de ser deslocado um byte para baixo na memória. X é o apontador da posição em que nos encontramos dentro da listagem BASIC. Ele apontava para o espaço que foi apagado, mas o byte seguinte da linha acaba de ser transferido para o mesmo local. Para considerar esse byte, temos que examinar a posição de memória novamente, em vez de observarmos o byte seguinte. Quando **BRA LTWO** volta ao princípio novamente, a primeira instrução incrementa o registro X, mudando para o próximo byte.

COMO PROCURAR CORDÕES

Se o programa não encontrar um caractere de espaço na instrução **CMPA #32**, a instrução **BNE LFIV** desviará para uma pequena rotina que procura aspas. **CMPA #34** compara o conteúdo do acumulador com o código ASCII das aspas. Se um caractere de aspas for encontrado, **CMPA #34** resultará 0; assim, a condição necessária à execução de **BNE LTWE** não será satisfeita e **LDB ICOM, PCR** carregará o registro B com o conteúdo da posição de memória onde os sinalizadores foram guardados.

EORB #1 executa a operação lógica "ou exclusivo" entre B e 1. Essa operação existe também em BASIC. **STB ICOM, PCR** coloca o resultado de volta no byte de sinalizadores; o bit zero é ativado como sinal de que um caractere de aspas foi encontrado. Se esse bit **ICOM** já era igual a 1, **EORB #1** retornará o valor 0: se o bit era 0, o resultado será 1. Assim, se o bit zero estava ligado, essa operação o desligará. Caso contrário, ela o ligará.

A FUNÇÃO DAS ASPAS

Se observarmos agora como as aspas envolvem as mensagens dentro de um programa, veremos que um cordão começa depois de um número ímpar de aspas e termina após um número par. Alternando o bit zero do byte de sinalizadores toda vez que um caractere de aspas for encontrado, o comando **EOR** deixará o bit valendo 1 após um número ímpar de aspas e 0 após um número par. O sinalizador ligado indica, portanto, que o programa se encontra em um cordão; desligado, ele revela que não se trata de um cordão. Esta é exatamente a condição que é testada por **TST ICOM, PCT** seguido de **BNE LTWO**.

Uma vez alterada a condição do sinalizador de aspas, **BRA LTWO** retorna ao princípio novamente, passando ao próximo byte da listagem BASIC.

Se o byte não estivesse entre aspas, a condição necessária de **BNE LTWE** teria sido satisfeita, e o programa prosseguiria com novas verificações.

PROCURANDO LINHAS DATA

As linhas **DATA** podem conter cordões; de forma que é melhor não comprimi-las também.

A instrução **CMPA #134** verifica se o byte no acumulador é o código do comando **DATA**. Caso seja, a condição necessária de **BNE LSIX** não será satisfeita.

Contudo, há outra situação em que o código 134 pode aparecer. A função **LOG** tem o código FF 86, em hexadecimal, ou 255 seguido de 134, em decimal. Assim, é preciso verificar se o byte anterior não é 255.

Como você deve estar lembrado, **LDA ,X+** incrementa o registro X após colocar nele o valor do acumulador. Desse modo, o registro X aponta agora para o byte posterior ao que está sendo considerado. Para verificar o byte anterior, o acumulador deve ser carregado com o conteúdo da memória cujo endereço é o resultado de X menos 2. A instrução **LDA −2,X** faz isso.

O comando **CMPA #255** verifica então se o byte anterior é 255. Se FF for encontrado, **BEQ LTWO** retornará ao princípio.

Se FF não for encontrado, isso significa que $86 é o código de um comando **DATA** e que, portanto, um sinalizador em **ICOM** deve ser ativado. Obviamente, não se trata do bit zero, que conta as aspas.

Assim, a instrução **LDB ICOM, PCR** coloca o valor de **ICOM** em B. Por sua vez, **EORB #2** realiza um "ou exclusivo" (**XOR**) entre B e 2. Isso ativa o bit dois.

COMO PROCURAR LINHAS REM

CMPA #130 verifica se o próximo byte é o código de uma instrução **REM**. Se for, **BEQ LSEV** levará o microprocessador ao rótulo **LSEV**. Se não for, **CMPA #131** verificará se o byte é um apóstrofo que funciona como **REM**.

Se, ao contrário, nenhum desses códigos for encontrado, a instrução **BNE LTWO** voltará ao princípio, passando ao próximo byte.

Caso um desses sinais seja encontrado, o programa deve verificar se não se trata de outras funções, cujos códigos são precedidos de FF. Isto é feito pelas três instruções seguintes: **LDA −2,X**, **CMPA #255** e **BEQ LTWO**.

Depois disso, o registro D é carregado com o valor de U, que corresponde ao endereço da próxima linha BASIC.

Por seu turno, a instrução **LEAX −1,X** decrementa o registro X, fazendo com que o apontador indique a posição do comando **REM** ou do apóstofro (nosso objetivo é apagar o trecho que vai desse ponto até o fim da linha). A seguir, **PSHS** guarda X e U na pilha e **SUBD ,S ++** subtrai o valor dos dois últimos bytes da pilha do registro D; o resultado é colocado em D mesmo. A

instrução também incrementa em duas vezes o apontador da pilha.

Como você já deve saber, o conteúdo de U é guardado antes de X na pilha. **SUBD ,S++** subtrai do valor de D a última coisa que foi guardada na pilha da máquina.

O sinal "++" após S incrementa o registro S (apontador da pilha da máquina) em duas vezes, ou seja, soma 2 a ele. Isso retira o valor de X — dois bytes — da pilha.

TFR X,Y transfere (copia) o conteúdo de X para Y. O registro Y não é usado para mais nada nesse programa, servindo apenas como um registro temporário para guardar o valor de X.

LDX ,U coloca o valor de U em X. U contém o endereço inicial da próxima linha. **LEAX −1,X** decrementa esse valor, de modo a fazê-lo corresponder ao endereço final da linha atual. **PSHS D,X** guarda o conteúdo desses registros na pilha para que possam ser usados pela rotina **SHIFT**.

TFR Y,D transfere para D o valor de Y. Assim, a posição do código da instrução **REM** está agora em D. **SUBD 4,S** subtrai de D o valor correspondente aos dois bytes que se encontram a partir do quinto byte da pilha. (S aponta para o primeiro byte da pilha. A expressão 4,S soma 4 a S, de forma que este passa a apontar para o quinto byte). Observe o que foi feito antes: esse valor é o antigo conteúdo do registro U, endereço inicial da próxima linha do programa BASIC.

Isso significa que o endereço inicial da linha é subtraído do endereço do código da instrução **REM**. O resultado nos dá o número de bytes entre o início da linha e a instrução. O que precisamos saber é se a instrução **REM** está no começo ou no meio da linha. No primeiro caso, podemos apagar também o número da linha. Isso é verificado por **CMPB #4**.

Se a instrução **REM** não estiver no começo da linha, **BNE LNIN** irá diretamente à instrução que chama a sub-rotina **SHIFT**. Contudo, se a **REM** estiver no início da linha, o resultado da operação será 4 e o último valor colocado na pilha será recuperado no registro D por **PULS D**. Soma-se 4 a D para que os quatro bytes iniciais da linha **REM** — correspondentes ao endereço inicial da linha seguinte e ao número da linha — também sejam apagados no processo de compressão. Finalmente, o conteúdo de D é devolvido à pilha.

A sub-rotina **SHIFT** é chamada por **BSRSHIFT** e o conteúdo original dos registros é recuperado da pilha. **LBRA LONE** volta ao início do programa.

Um desvio longo — **L**ong **BRA**nch — é usado porque o salto é maior que 128 bytes.

A ROTINA DE COMPRESSÃO

O comando **LDD ,U** coloca em D o conteúdo do endereço apontado por U — em outras palavras, D vai conter o endereço inicial da próxima linha do programa em BASIC. Se esse valor for zero, teremos chegado ao fim do programa e a instrução **BEQ LTWO** sairá do laço principal do programa.

Se não se tratar do fim do programa em BASIC, o mesmo valor será colocado em X por **LDX ,U**. O valor correspondente aos dois bytes localizados a partir do terceiro byte da pilha será então subtraído por **SUBD 2,S** do endereço em D (endereço inicial da próxima linha).

Por outro lado, devemos ter em mente que estamos no meio de uma sub-rotina e os dois bytes correspondentes ao endereço de retorno da sub-rotina foram colocados na pilha da máquina — daí a necessidade do 2,S.

O resultado dessa operação é o endereço inicial que a próxima linha BASIC ocupará, uma vez feita a compressão. Ela é colocada na posição de memória apontada por U, que corresponde aos dois primeiros bytes da linha BASIC. Estes, por sua vez, indicam o endereço da próxima linha. Em outras palavras, estamos fazendo com que esse valor coincida com as mudanças de endereço causadas pela compressão.

O valor de X que corresponde ao endereço inicial antigo é colocado em U e **BRA SHIFT** volta novamente ao início da sub-rotina **SHIFT**.

A única saída desse laço é o **BEQ**, que só será executado quando todo o programa BASIC tiver passado por esse tipo de modificação. Quando a preparação for finalmente completada, podemos comprimir nosso programa. O comando **LDD 4,S** transporta para D o valor de X guardado na pilha, isto é, ele coloca o endereço final da linha atual se estivermos eliminando um comentário **REM**, ou a posição atual na linha, caso estejamos eliminando um espaço.

SUBD 2,S subtrai o número de bytes que serão eliminados.

O resultado é a posição que o próximo byte da linha deve ocupar; ele é colocado em U pela instrução **TFR D,U**.

LDX 4,S recupera outra vez o antigo valor de X guardado na pilha. Assim, o endereço antigo do próximo byte está em X e o novo em U.

LDA ,X+ coloca o byte apontado por X em A e incrementa o valor de X. **STA ,U+** põe o mesmo valor na posição de memória apontada por U e incrementa o valor de U. Assim, o próximo byte do programa não comprimido é transferido para sua nova posição, no programa comprimido. Além disso, os dois apontadores X e U são incrementados para que apontem para o próximo byte e para a próxima posição.

CMPX #27 compara o conteúdo de X com o conteúdo da variável do sistema que fica nos endereços 27 e 28. Ela corresponde ao endereço inicial da área das variáveis, que deve ficar logo acima do final do programa em BASIC. Essa instrução verifica se chegamos ao final da listagem. Se isto ainda não aconteceu, **BLO** — desvio se for menor — desviará para o rótulo **SHTHR**, continuando a compressão.

Quando todo o programa tiver sido comprimido, o microprocessador executará **LDD 27**. Essa instrução coloca o apontador do final do programa no registro D. **SUB 2,S** subtrai o número de bytes eliminados e **STD 27** coloca o apontador atualizado de volta em seu lugar. **RTS** retorna à rotina principal.

COMO USAR O COMPRESSOR

Esse programa pode ser montado com qualquer endereço inicial, desde que o local tenha sido protegido. Depois de gravar o programa-fonte e montar o compressor, podemos usar **NEW** para eliminar o Assembler e então carregar o programa que vamos compactar. Para executar a compressão, use:

```
DEF USR0 = endereço inicial
```

onde o endereço inicial é a origem. Depois, digite:

```
PRINT USR0(0)
```

Devemos, contudo, ser muito cuidadosos. Não será possível editar o programa depois da compressão; portanto, ele deve ter sido exaustivamente testado. Muita atenção com os comandos **GOTO** e **GOSUB** que se referiam a linhas **REM**: eles devem ser mudados.

Para gravar o compressor em código, digite:

```
CSAVEM "CMPRSS" endereço
inicial, endereço final,
endereço inicial
```

Para carregar o programa de volta, use:

```
CLOADM "CMPRSS"
```

LINHA	FABRICANTE	MODELO
Apple II +	Appletronica	Thor 2010
Apple II +	CCE	MC-4000 Exato
Apple II +	CPA	Absolutus
Apple II +	CPA	Polaris
Apple II +	Digitus	DGT-AP
Apple II +	Dismac	D-8100
Apple II +	ENIAC	ENIAC II
Apple II +	Franklin	Franklin
Apple II +	Houston	Houston AP
Apple II +	Magnex	DM II
Apple II +	Maxitronica	MX-2001
Apple II +	Maxitronica	MX-48
Apple II +	Maxitronica	MX-64
Apple II +	Maxitronica	Maxitronic I
Apple II +	Microcraft	Craft II Plus
Apple II +	Milmar	Apple II Plus
Apple II +	Milmar	Apple Master
Apple II +	Milmar	Apple Senior
Apple II +	Omega	MC-400
Apple II +	Polymax	Maxxi
Apple II +	Polymax	Poly Plus
Apple II +	Spectrum	Microengenho I
Apple II +	Spectrum	Spectrum ed
Apple II +	Suporte	Venus II
Apple II +	Sycomig	SIC I
Apple II +	Unitron	AP II
Apple II +	Victor do Brasil	Elppa II Plus
Apple II +	Victor do Brasil	Elppa Jr.
Apple IIe	Microcraft	Craft IIe
Apple IIe	Microdigital	TK-3000 IIe
Apple IIe	Spectrum	Microengenho II
MSX	Gradiente	Expert GPC-1
MSX	Sharp	Hotbit HB-8000
Sinclair Spectrum	Microdigital	TK-90X
Sinclair Spectrum	Timex	Timex 2000
Sinclair ZX-81	Apply	Apply 300
Sinclair ZX-81	Engebras	AS-1000
Sinclair ZX-81	Filcres	NEZ-8000
Sinclair ZX-81	Microdigital	TK-82C
Sinclair ZX-81	Microdigital	TK-83
Sinclair ZX-81	Microdigital	TK-85
Sinclair ZX-81	Prologica	CP-200
Sinclair ZX-81	Ritas	Ringo R-470
Sinclair ZX-81	Timex	Timex 1000
Sinclair ZX-81	Timex	Timex 1500
TRS-80 Mod. I	Dismac	D-8000
TRS-80 Mod. I	Dismac	D-8001/2
TRS-80 Mod. I	LNW	LNW-80
TRS-80 Mod. I	Video Genie	Video Genie I
TRS-80 Mod.III	Digitus	DGT-100
TRS-80 Mod.III	Digitus	DGT-1000
TRS-80 Mod.III	Kemitron	Naja 800
TRS-80 Mod.III	Prologica	CP-300
TRS-80 Mod.III	Prologica	CP-500
TRS-80 Mod.III	Sysdata	Sysdata III
TRS-80 Mod.III	Sysdata	Sysdata Jr.
TRS-80 Mod.IV	Multix	MX-Compacto
TRS-80 Mod.IV	Sysdata	Sysdata IV
TRS-Color	Codimex	CS-6508
TRS-Color	Dynacom	MX-1600
TRS-Color	LZ	Color 64
TRS-Color	Microdigital	TKS-800
TRS-Color	Prologica	CP-400

FABRICANTE	MODELO	PAÍS	LINHA
Appletronica	Thor 2010	Brasil	Apple II +
Apply	Apply 300	Brasil	Sinclair ZX-81
CCE	MC-4000 Exato	Brasil	Apple II +
CPA	Absolutus	Brasil	Apple II +
CPA	Polaris	Brasil	Apple II +
Codimex	CS-6508	Brasil	TRS-Color
Digitus	DGT-100	Brasil	TRS-80 Mod.III
Digitus	DGT-1000	Brasil	TRS-80 Mod.III
Digitus	DGT-AP	Brasil	Apple II +
Dismac	D-8000	Brasil	TRS-80 Mod. I
Dismac	D-8001/2	Brasil	TRS-80 Mod. I
Dismac	D-8100	Brasil	Apple II +
Dynacom	MX-1600	Brasil	TRS-Color
ENIAC	ENIAC II	Brasil	Apple II +
Engebras	AS-1000	Brasil	Sinclair ZX-81
Filcres	NEZ-8000	Brasil	Sinclair ZX-81
Franklin	Franklin	USA	Apple II +
Gradiente	Expert GPC1	Brasil	MSX
Houston	Houston AP	Brasil	Apple II +
Kemitron	Naja 800	Brasil	TRS-80 Mod.III
LNW	LNW-80	USA	TRS-80 Mod. I
LZ	Color 64	Brasil	TRS-Color
Magnex	DM II	Brasil	Apple II +
Maxitronica	MX-2001	Brasil	Apple II +
Maxitronica	MX-48	Brasil	Apple II +
Maxitronica	MX-64	Brasil	Apple II +
Maxitronica	Maxitronic I	Brasil	Apple II +
Microcraft	Craft II Plus	Brasil	Apple II +
Microcraft	Craft IIe	Brasil	Apple IIe
Microdigital	TK-3000 IIe	Brasil	Apple IIe
Microdigital	TK-82C	Brasil	Sinclair ZX-81
Microdigital	TK-83	Brasil	Sinclair ZX-81
Microdigital	TK-85	Brasil	Sinclair ZX-81
Microdigital	TK-90X	Brasil	Sinclair Spectrum
Microdigital	TKS-800	Brasil	TRS-Color
Milmar	Apple II Plus	Brasil	Apple II +
Milmar	Apple Master	Brasil	Apple II +
Milmar	Apple Senior	Brasil	Apple II +
Multix	MX-Compacto	Brasil	TRS-80 Mod.IV
Omega	MC-400	Brasil	Apple II +
Polymax	Maxxi	Brasil	Apple II +
Polymax	Poly Plus	Brasil	Apple II +
Prologica	CP-200	Brasil	Sinclair ZX-81
Prologica	CP-300	Brasil	TRS-80 Mod.III
Prologica	CP-400	Brasil	TRS-Color
Prologica	CP-500	Brasil	TRS-80 Mod.III
Ritas	Ringo R-470	Brasil	Sinclair ZX-81
Sharp	Hotbit HB-8000	Brasil	MSX
Spectrum	Microengenho I	Brasil	Apple II +
Spectrum	Microengenho II	Brasil	Apple IIe
Spectrum	Spectrum ed	Brasil	Apple II +
Suporte	Venus II	Brasil	Apple II +
Sycomig	SIC I	Brasil	Apple II +
Sysdata	Sysdata III	Brasil	TRS-80 Mod.III
Sysdata	Sysdata IV	Brasil	TRS-80 Mod.IV
Sysdata	Sysdata Jr.	Brasil	TRS-80 Mod.III
Timex	Timex 1000	USA	Sinclair ZX-81
Timex	Timex 1500	USA	Sinclair ZX-81
Timex	Timex 2000	USA	Sinclair Spectrum
Unitron	AP II	Brasil	Apple II +
Victor do Brasil	Elppa II Plus	Brasil	Apple II +
Victor do Brasil	Elppa Jr.	Brasil	Apple II +
Video Genie	Video Genie I	USA	TRS-80 Mod. I

INPUT foi especialmente projetado para microcomputadores compatíveis com as sete principais linhas existentes no mercado.
Os blocos de textos e listagens de programas aplicados apenas a determinadas linhas de micros podem ser identificados por meio dos seguintes símbolos:


Sinclair ZX-81


TRS-80


TK-2000


MSX


Spectrum


TRS-Color


Apple II

Quando o emblema for seguido de uma faixa, então tanto o texto como os programas que se seguem passam a ser específicos para a linha indicada.

NO PRÓXIMO NÚMERO

PROGRAMAÇÃO DE JOGOS
Você quer aprender a pilotar aviões? Eis o começo de um programa de vôo simulado.

PROGRAMAÇÃO BASIC
Teclas programáveis: um recurso fantástico para quem quer escrever programas "profissionais" no MSX.

APLICAÇÕES
Você não precisa ser um Guimarães Rosa, para escrever bons textos no computador. Comece fazendo rascunhos.

PROGRAMAÇÃO BASIC
O que é um desenho do tipo *wireframe*. Organize as rotinas.